Universidad de Alcalá
Escuela Politécnica Superior

INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

Análisis de la accesibilidad en el desarrollo de videojuegos en entornos 3D con prototipo en Unreal Engine

Autor: Adrián de Juan Lora

Tutores: José María Gutiérrez Martínez
Juan Aguado Delgado

2019

Palabras Clave:

Videojuegos, 3D, Accesibilidad, Unreal Engine.

Keywords:

Videogames, 3D, Accessibility, Unreal Engine.

Resumen corto:

El objetivo de este proyecto es elaborar una rúbrica que permita medir el grado de accesibilidad de un videojuego. Para ello, se analizará el estado actual de la accesibilidad en los videojuegos como producto software, así como las diferentes técnicas utilizadas para hacer accesible un videojuego 3D y los obstáculos que surgen en el desarrollo de éste. También se ha realizado un pequeño prototipo con el motor Unreal Engine para ver la viabilidad real de un videojuego accesible.

Short summary:

The objective of this project is to make a rubric that allows to measure how accessible a game is. In order to do so, we will study the actual accessibility state in videogames as a software product and the different techniques used to make a 3D videogame accessible and the obstacles that arise during its development. Also, it has been realized a small prototype using the Unreal Engine technology.

Agradecimientos.

A mis profesores del grado de ingeniería informática, que me han ayudado a comprender y formar parte de este mundo.

A mis padres, por acompañarme a lo largo de toda mi vida y guiarme en el camino.

A Jorge, por enseñarme a ser siempre leal.

A Renata, por ayudarme siempre, dejándome apoyarme en su alma.

A Nacho, Mario y Bea, sin ellos, jamás habría llegado hasta aquí.

A Irene, siempre estás en mi memoria.

A Samuel, junto a él, da igual que llegue el fin del mundo, nos pillará bailando.

Índice:

1. Introducción.

Una de las características más importantes de cualquier producto software es la accesibilidad. Hacer posible que los distintos perfiles de diversidad funcional puedan usar de manera cómoda y sencilla nuestro producto. La accesibilidad suele verse perjudicada por muchos obstáculos en los productos software. Sin embargo, presenta aún más obstáculos en los videojuegos, particularmente en los videojuegos 3D.

Los videojuegos 3D son los productos más vendidos dentro de la industria de los videojuegos. Por ejemplo, en 2017, los 10 videojuegos más vendidos contaban todos con un motor 3D [1]. Sin embargo, prácticamente todos esos videojuegos contaban con muy pocas opciones de accesibilidad.

Hay muchas barreras que hoy en día no son salvables, ya que no se dispone del suficiente estudio o tecnología para poder superarlas. Sin embargo, en su gran parte, las barreras de accesibilidad son evitables si se pone énfasis en dicha característica durante el diseño y el desarrollo del videojuego.

Se espera que este proyecto pueda ser usado como una guía para las distintas consideraciones que se han de tomar durante el desarrollo de un videojuego o cualquier producto software, pudiendo así mejorar la accesibilidad de éste.

2. Objetivo.

El objetivo principal de este proyecto es el desarrollo de una rúbrica que sirva para medir el nivel de accesibilidad de un videojuego 3D.

Los objetivos secundarios de este proyecto son los siguientes:

- Obtener un sistema de puntuación en la rúbrica.
- Obtener un valor total de puntuación en la rúbrica y tramos en la puntuación que den una idea aproximada del desarrollo de la accesibilidad en un videojuego.
- Adquirir conocimientos sobre accesibilidad en productos software.
- Adquirir conocimientos sobre desarrollo de videojuegos 3D con la herramienta Unreal Engine.
- Elaborar un prototipo de videojuego accesible con la herramienta Unreal Engine.

3. Estado del Arte

El objetivo de esta sección del trabajo es hacer una investigación acerca de los diferentes perfiles de diversidad funcional que existen y que necesidades implican para los usuarios. También se investigará acerca de los motores de videojuegos como herramienta de trabajo. Después, se expondrá el concepto de accesibilidad actual, así como las soluciones ya existentes en cualquier medio para hacer accesible un producto software. Por último, se hará una valoración de los apartados anteriores.

a. Perfiles de diversidad funcional.

En este apartado, se van a mostrar y desarrollar los distintos perfiles de diversidad funcional. La Cruz Roja divide los tipos de Discapacidad (De aquí en adelante, Diversidad Funcional) en tres. Física o motora, Mental o cognitiva y sensorial [2].

La diversidad funcional motora implica trastornos físicos como el Parkinson que impiden el movimiento de la persona, la cognitiva refleja problemas de comunicación o aprendizaje y la sensorial se divide en dos, auditiva y visual.

i. Sensorial

La diversidad funcional sensorial es aquella que afecta a los sentidos de la audición y la vista. En determinadas ocasiones, también se aplica al tacto, gusto y olfato, pero en cuanto se refiere a accesibilidad en productos de entretenimiento, los mayores actores, y el foco de la accesibilidad, son los dos primeros.

Auditiva

Existen varios tipos de diversidad funcional auditiva, si bien hay 3 perfiles en particular que afectan a un mayor número de personas. Estas enfermedades son la Hipoacusia, la cofosis y la Hiperacusia.

Hipoacusia y Cofosis

La pérdida auditiva puede ser parcial (hipoacusia) o total (cofosis). Dependiendo del grado de profundidad de esta. La hipoacusia, comúnmente denominada sordera puede venir dada por muchos factores, como enfermedades en la infancia o defectos congénitos. Dependiendo de que cause la pérdida auditiva en el oído, la CDC diferencia varios tipos [3]:

- Pérdida auditiva conductiva. Ocurre cuando un bloqueo impide el paso del sonido del odio externo al medio. Normalmente es tratable mediante medicamentos y/o cirugía.
- Pérdida auditiva neurosensorial. Ocurre cuando existe alguna alteración en el funcionamiento del oído interno o en el nervio auditivo.
- Pérdida auditiva mixta. Es una combinación de las dos anteriores.

- Trastorno del espectro de neuropatía auditiva. Ocurre cuando el sonido entra de forma normal al oído, pero no se transmite de manera que el cerebro lo pueda interpretar.

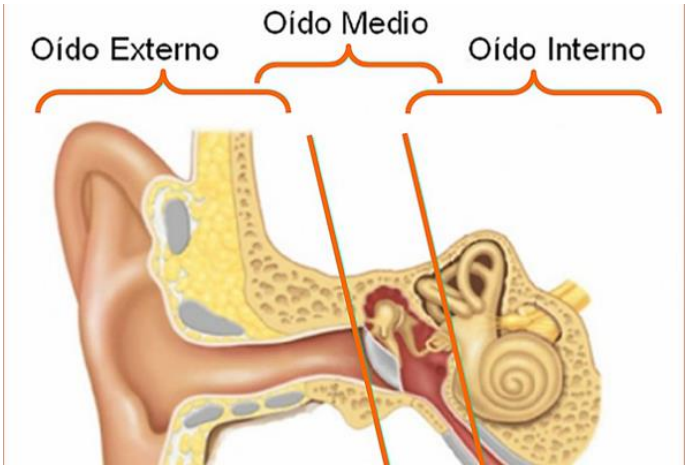

Fig. 1 Divisiones del Oído. [F1]

La *hipoacusia* se puede dar en cuatro casos dependiendo de su gravedad: Leve, Moderada, Grave y Profunda. La diferencia principal se encuentra en el volumen de decibelios que el oído es capaz de procesar.

La pérdida de la audición puede afectar a uno o varios oídos, en misma o distinta manera. También suele afectar al lenguaje de la persona.

La OMS cifra la cantidad de personas que sufren *hipoacusia* o *cofosis* en 466 millones [4].

Hiperacusia.

La *hiperacusia* o *algiacusia* es un síndrome que se caracteriza por una hipersensibilidad auditiva creando intolerancia a la mayoría de los sonidos cotidianos que rodean a la persona. En casos extremos, incluso sonidos que no somos capaces de percibir de manera habitual pueden llegar a dañar a la persona. Existen dos tipos de *hiperacusia* la *coclear*, que es la más común o la *vestibular*

Los síntomas de la *hiperacusia coclear* se caracterizan por dolores de oído, ataques de pánico, episodios de llanto y *acúfenos* (sonidos percibidos que no provienen del exterior, sino que se generan dentro del oído de la persona).

Los síntomas de la *hiperacusia vestibular* son más difíciles de identificar, ya que suelen ser exclusivamente mareos y vértigos cuando se perciben sonidos fuertes o desagradables.

Visual

Los perfiles de diversidad funcional visual son aquellos que sufren de alguna dificultad en el sentido de la vista, ya sea la ausencia total, o distintos grados de pérdida y/o deformación de este sentido. Las principales enfermedades que afectan a este sentido son la ceguera, la baja visión y el Daltonismo.

Baja visión y ceguera

La baja visión es una condición que afecta a la vista de la persona. En casos extremos llega a la ceguera.

La baja visión se caracteriza por la dificultad de la persona para distinguir formas y colores en su entorno, de cerca o de lejos. Englobando así condiciones como la *miopía* o la *hipermetropía*. Si se diagnostica a una temprana edad, se pueden tomar medidas para su prevención e incluso mejora con el tiempo. La ceguera es la incapacidad total de percibir el entorno, normalmente debido a la incapacidad de percibir la luz a través del ojo.

La OMS estima que el 80% de todos los casos de baja visión son evitables [5]. Y estima la cifra de personas afectadas por esta condición en 1300 millones.

La baja visión puede venir dada por varias causas:

- Cataratas: Es una patología congénita y pediátrica que describe el ennegrecimiento de las lentes cristalinas, impidiendo estas el paso correcto de la luz.
- Glaucoma: Es una enfermedad congénita y pediátrica ocular caracterizada por un incremento.
- Infecciones
- Lesiones oculares
- Errores de refracción: Como la *miopía* o la *hipermetropía*. Pueden llegar a ser degenerativos.
- Diabetes.

Daltonismo.

El daltonismo, o *discromatopsia congénita es* una alteración en la percepción de los colores. Su nombre viene de John Dalton, su descubridor. El Daltonismo es un término muy amplio en su significado y por ello se distinguen varios tipos dependiendo de en qué grado o que colores la persona es capaz de observar. Así, se distinguen:

- Acromático: El Daltonismo acromático o *Acromatopsia* es aquel en el que el individuo solo puede ver en una escala de grises. Es decir, distingue variaciones en la luz, pero solo ve en blanco y negro, pues no posee ninguno de los tres tipos de conos. Es una condición muy poco frecuente que se presenta únicamente en 1 de cada 100 000 personas.
- Monocromático: El Daltonismo monocromático es similar al Acromático, ya que la persona distingue variaciones en la luz y en el color. Sin embargo, la persona posee uno de los tres pigmentos de los conos (Rojo, Azul o verde) y observa todo en una escala de ese color.

- Dicromático: El Daltonismo dicromático o *dicromatismo* es una condición por la cual uno de los fotorreceptores retinianos del color sufre una disfunción, la condición es hereditaria y se presenta en tres variantes.

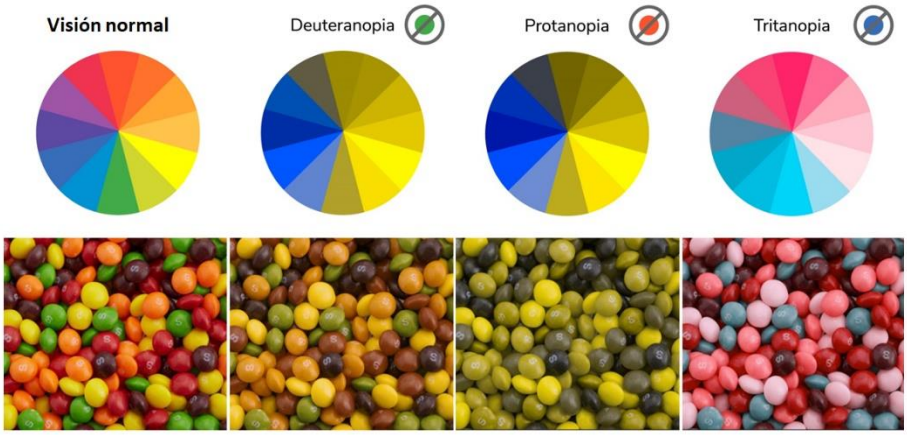

Fig. 2 Simulación de los diferentes tipos de Daltonismo[F2]

  - Deuteranopia: Cuando están ausentes los fotorreceptores retinianos del color verde
  - Protanopia: Cuando están ausentes los fotorreceptores retinianos del color rojo
  - Tritanopia: Es la menos común, sucede cuando están ausentes los fotorreceptores retinianos del color azul.
  -

- Tricromático anómalo: Es el grupo más abundante y común de los daltónicos, ocurre cuando la persona posee los tres tipos de conos, pero con modificaciones funcionales, que le llevan a confundir un color con otro.
  - Deuteranomalía: Es la más frecuente, la persona tiene dificultades distinguiendo el verde.
  - Protanomalía: La persona tiene dificultades distinguiendo el rojo.
  - Tritanomalía: Es la menos común, la persona tiene dificultades distinguiendo el azul.

ii. Cognitiva.

Existen varias enfermedades que afectan al grado cognitivo de la persona. La diversidad funcional cognitiva se caracteriza por una dificultad en el procesamiento y el manejo de la información, afectando así, principalmente a la memoria y a los sentidos. Algunos trastornos cognitivos, como la epilepsia, pueden desencadenar comportamientos físicos, en el caso de la epilepsia, convulsiones.

El síndrome de Down y la Dislexia son dos de las enfermedades más comunes durante el desarrollo cognitivo de la persona.

Síndrome de Down.

El síndrome de Down es un trastorno genético debido a la presencia de una copia extra del cromosoma 21.

Las causas del síndrome de Down son desconocidas, aunque se le ha intentado relacionar estadísticamente con una edad materna avanzada o problemas genéticos en los progenitores, si bien no hay nada demostrado.

El síndrome de Down representa el 25% de los casos de discapacidad cognitiva, ya que, según las Naciones Unidas, se estima a nivel mundial una incidencia entre 1 de cada 1000 y 1 de cada 1100 recién nacidos [6].

El síndrome de Down presenta características físicas reconocibles, que pueden ayudar en su diagnóstico. Las personas con Síndrome de Down suelen presentar una propensión a los problemas oculares y auditivos, así como problemas intestinales y de sobrepeso.

Fig. 3 Variación de los atributos físicos del Síndrome de Down con la edad.[F3]

Actualmente no existe ningún tratamiento para el Síndrome de Down, sin embargo, estos problemas derivados de él pueden tratarse para hacerlos más leves. El síndrome de Down dificulta también la adquisición de nueva información, dificultando la educación de la persona., si bien esta discapacidad cognitiva puede variar según la persona.

Dislexia.

La dislexia es una dificultad al aprendizaje que afecta tanto a la lectura como a la escritura. Se presenta en durante la infancia, aumentando sus síntomas con la edad. Empezando con problemas en la dicción y en el habla, la asignación de fonemas erróneos en la lectura y siguiendo con dificultad para distinguir números, letras y comprender textos largos.

La dislexia se causa por una dificultad en la automatización de los procesos de comprensión. Las personas disléxicas también pueden mostrar problemas siguiendo órdenes o conjuntos de órdenes muy complicados.

La dislexia puede tratarse, especialmente durante la infancia, ayudando al niño a mejorar en la automatización de dichos procesos. Existen varias terapias con mayor o menor porcentaje de éxito, sin embargo, no hay un tratamiento definido.

La dislexia es la Dificultad Especifica del Aprendizaje (DEA) más común en el mundo, y se estima que un 10% de la población sufre dislexia. [7]

iii. Motora.

La diversidad motora se define como la falta de un miembro o de su funcionalidad. Sin embargo, el tipo de discapacidad que se sufre en particular se clasifica siguiendo 4 criterios distintos según la cruz roja [8]. Además de dichos criterios, hay que tener en cuenta

enfermedades degenerativas como el Parkinson o problemas nerviosos que pueden afectar la movilidad fina de la persona.

Según el tipo.

Esta clasificación distingue en función del tipo de síntomas que se muestren.

- **Espasticidad:** cuando se sufre un aumento exagerado del tono muscular (hipertonía), por lo que se sufren movimientos exagerados y poco controlados.
- **Atetosis:** se pasa de hipertonía a hipotonía, por lo que hay movimientos incoordinados, lentos, no controlables. Estos movimientos afectan a las extremidades y en algunos casos los músculos de la cara y la lengua, lo que provoca hacer muecas o salivar. Pueden tener problemas para coordinar los movimientos musculares necesarios para el habla (disartria).
- **Ataxia:** sentido defectuoso de la marcha y descoordinación motora tanto fina como gruesa. Es una forma rara en la que las personas afectadas caminan inestablemente, poniendo los pies muy separados uno del otro
- **Mixto:** es lo más frecuente, manifiestan diferentes características de los anteriores tipos. La combinación más frecuente es la de espasticidad y movimientos atetoides.

Según el tono.

Esta clasificación distingue en función de los cambios en la tonalidad de la musculación.

- **Isotónico:** tono normal.
- **Hipertónico:** aumento del tono.
- **Hipotónico:** tono disminuido.

Según la parte del cuerpo afectada.

Esta clasificación distingue en función de cuantos y que miembros son afectados.

- **Hemiplejia o Hemiparesia:** se encuentra afectada uno de los dos lados del cuerpo.
- **Diplejía o diparesia:** la mitad inferior está más afectada que la superior.
- **Cuadriplejía o cuadriparesia:** los cuatro miembros están paralizados.
- **Paraplejía o Paraparesia:** afectación de los miembros inferiores.
- **Monoplejía o monoparesia:** se encuentra afectado un sólo miembro.
- **Triplejía o triparesia:** se encuentran afectados tres miembros.

Según el grado de afectación.

Esta clasificación distingue en función de la autonomía que presente el miembro.

- **Grave:** no hay prácticamente autonomía.
- **Moderada:** tiene autonomía o necesita alguna ayuda asistente.
- **Leve:** tiene total autonomía.

b. Concepto de motor de videojuegos.

Desarrollar un videojuego no es tarea sencilla. Como cualquier producto software, se requieren tareas de análisis, diseño e implementación, pero todas ellas requieren un gran esfuerzo y carga de trabajo.

Esto se debe a que, en un videojuego, normalmente, las funcionalidades (comúnmente llamadas mecánicas) a desarrollar no son sencillas. Como ejemplo, la mecánica de salto en cualquier juego de Mario. El salto de Mario, pulsando un solo botón, cambiará en función de cuánto tiempo se deje pulsado el botón, si se está encadenando con un salto anterior, si Mario está corriendo, está quieto, está agachado, etc. Desde hace un tiempo, se suele hacer referencia a esta condición con el problema de la puerta [9].

Fig. 4 Ilustración del problema de la puerta[F4]

Dado que se requiere tanta atención en otras partes del proyecto, es una práctica común hoy en día definir el funcionamiento base del videojuego con un motor. Este motor se encarga de las cosas más tediosas de manera automática. Con los Software Development Kits (SDKs) apropiados se puede programar el juego para distintas plataformas a la vez. El motor también se encarga de mantener un sistema de físicas que afectan al entorno de juego (gravedad, inercia, etc.)

No solo ayudan con temas de compatibilidad entre plataformas, compilación, exportación, etc. Un motor de videojuegos también dispone de herramientas para la creación de entornos, diseño de personajes, creación de clima, uso de texturas y otros varios problemas dedicados normalmente a un apartado artístico. Además, el motor dispone normalmente de herramientas que facilitan la programación de estas funciones, muchas veces con interfaces que permiten usar controles *drag and drop* o *sliders* etc. Esto se hace así para que personas con escasos conocimientos de programación, pero grandes conocimientos artísticos puedan participar fácilmente en el proceso de desarrollo.

Dicho de otra forma, un motor de videojuegos es un software que dispone de un conjunto de herramientas y características que facilitan el proceso de desarrollo de un videojuego.

En nuestro caso, vamos a usar el motor Unreal Engine, en su versión 4. El motor Unreal Engine nos interesa por que es uno de los más potentes y conocidos de la industria y su modelo de pago es interesante. Es de uso gratuito, sólo se debe pagar un 5% de los beneficios obtenidos con el videojuego si éste supera los 3.000 dólares en ganancias por producto y trimestre.

Unreal Engine está destinado al desarrollo de videojuegos 3D, si bien su versatilidad permite el desarrollo de varios tipos de videojuegos. Unreal Engine es un motor desarrollado y vendido por la compañía *Epic Games* que cuenta con varias opciones de accesibilidad propias.

c. Concepto de Accesibilidad y estado actual.

Somos conscientes de que un videojuego puede ser un producto artístico y el director creativo del juego puede crear los desafíos que éste ofrece de la manera que considere adecuada. Sin embargo, también consideramos importante que toda persona pueda jugar al juego como prefiera, cambiando los controles y la dificultad a lo que le parezca adecuado.

i. General

En este apartado se definen las opciones generales de accesibilidad, que, aunque no ayuden a un perfil en particular, son moderadamente útiles para todos.

Opciones.

La mejor manera de constituir una accesibilidad general es mediante el menú de opciones. Cuantas más opciones editables existan en el videojuego, como subtítulos, tamaño de texto, dificultad, controles diferentes, selecciones de volumen, etc. más fácilmente puede el jugador acomodar su uso a lo que personalmente necesita, y más accesible se vuelve el juego. Más adelante listaremos opciones que se deberían añadir específicamente para distintos tipos de diversidad funcional, como pueden ser los subtítulos o la edición de controles.

Sin embargo, existe un estigma en cuanto a añadir opciones, el diseñador de juego debería ser el que marcara la experiencia, especialmente si cree que es parte de la experiencia principal del juego, el jugador no debería decidir sobre ella ya que podría

desviarse de la intención del diseñador. Creemos que la solución para esto es la comunicación entre ambas partes.

Recientemente varios juegos han añadido este tipo de opciones que permiten cambiar la mecánica del juego. Consideramos el mejor ejemplo el videojuego *Celeste*. *Celeste* es un juego de plataformas que consta de gran dificultad y requiere movimiento y reflejos precisos. La dificultad del juego es un tema central del mismo, tanto narrativa como mecánicamente. Pero el juego te deja modificar numerosas opciones que afectan a esta dificultad. Para ello, se debe activar el modo asistido, al activarlo un mensaje de los desarrolladores salta explicando que este modo está pensado para aquellas personas que no estén disfrutando el juego debido a su dificultad. No es un modo fácil al que cambiar cuando pienses que el juego es difícil, es un modo que existe para aquellas personas que no podrían disfrutar el juego de otra forma. Y eso queda completamente claro.

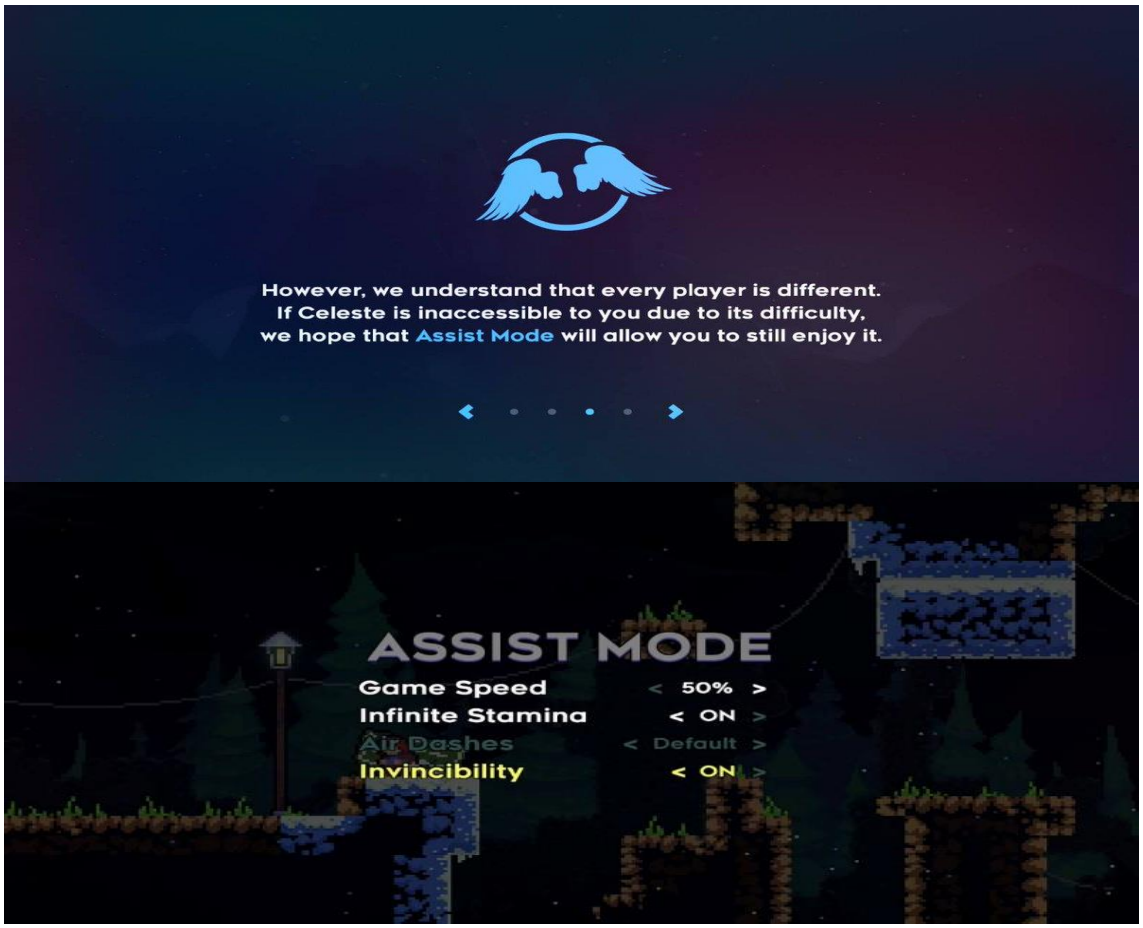

Fig. 5 Pantalla de configuración del modo Asistido de Celeste[F5]

ii. Sensorial

Las diversidades funcionales auditivas y visuales suelen englobarse dentro de lo conocido como diversidad sensorial. Sin embargo, las soluciones específicas de diseño que presentan estas formas de discapacidad son suficientes como para considerarlas separadas en dos subapartados distintos.

Auditivo

En este apartado se hablará de las principales soluciones de accesibilidad para personas con perfiles de diversidad funcional auditiva.

Subtítulos

Los subtítulos están bastante popularizados en el mundo de los videojuegos, más del 60% de los jugadores de *Assassins Creed Origins* usan subtítulos [10], sin embargo, no existe ningún estándar que permita desarrollarlos de manera correcta, por lo que muchos juegos implementan subtítulos que no son una ayuda adecuada. Los subtítulos deberían cumplir siempre ciertas condiciones:

- Los subtítulos han de ser grandes, dependiendo del juego y de la pantalla el tamaño del subtitulo puede variar. Una buena práctica es permitir al jugador cambiar el tamaño del texto. El tamaño mínimo recomendado es de 46 pixeles.
- Los subtítulos han de usar fuentes sencillas. Preferiblemente una Sans Serif, que permita leerlos de manera adecuada.
- Los subtítulos han de mostrar contraste contra el fondo. La manera más sencilla de poner esto es implementar un fondo translucido a los propios subtítulos. Si no fuera posible, una fuente de color blanco, con un borde negro grueso es también buena opción.
- Los subtítulos han de ser cortos. Es decir, han de mostrar pocas palabras a la vez. En sus normas de subtitulación Netflix y la BBC recomiendan un máximo de 37 caracteres por evento de subtitulo, y no más de 2 líneas como máximo. Los subtítulos deben durar al menos 0.3 segundos y 7 segundos como máximo. Las frases pueden ser separadas después de un signo de puntuación, antes de una conjunción o antes de una preposición.
- También es recomendable dejar algunos frames sin subtítulos antes de un nuevo evento de subtitulo. Esto ayuda al usuario a saber que un nuevo subtitulo acaba de aparecer en pantalla.
- Los subtítulos deberían indicar quien está hablando, especialmente si se muestran varios personajes en pantalla. El último videojuego de la saga *Tomb Raider* hace esto asignándole un color de texto diferente a cada personaje.
- Los subtítulos deberían estar presentes siempre. No solo en las escenas, sino también para las conversaciones y el ruido de fondo. La empresa Valve suele implementar mediante una opción en el menú la posibilidad de alternar entre subtítulos parciales (voces y conversaciones) y subtítulos totales (Ruidos ambientales, onomatopeyas, sonidos además de lo ya implementado por los parciales.)

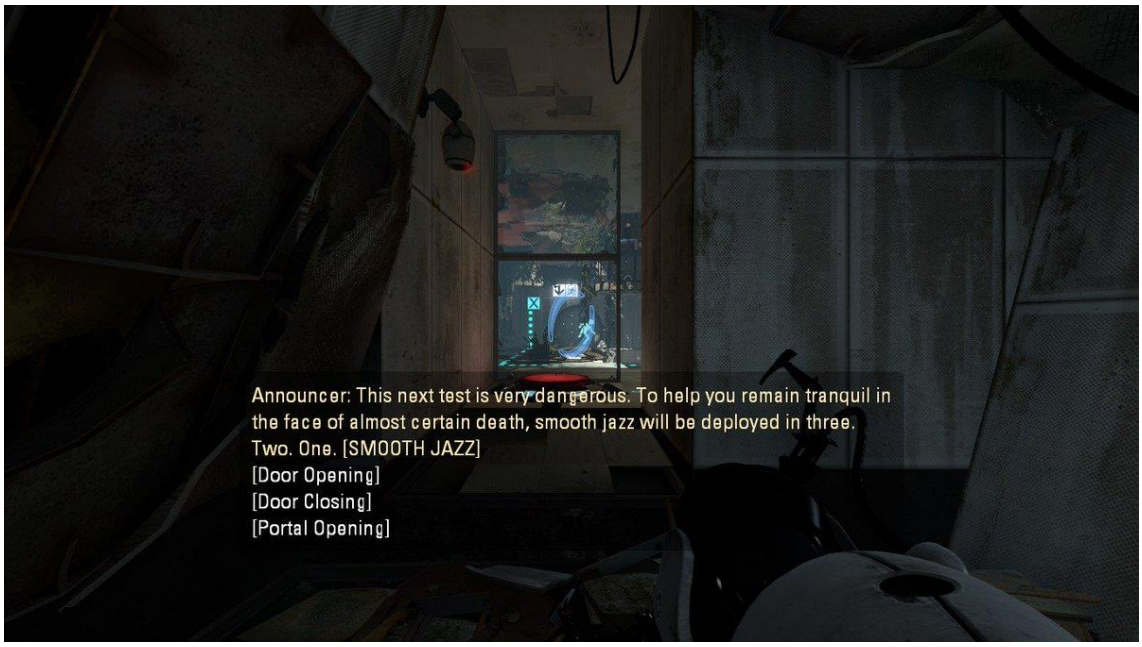

Fig. 6 Subtítulos Totales en Portal 2[F6]

- Los subtítulos deben tener contraste contra cualquier fondo. Poner un trasfondo traslucido en un color oscuro contra un subtítulo blanco o amarillo suele cumplir esta función, también se puede usar un borde negro y grueso que rodee al subtitulo.

Pistas visuales.

La mayoría de los videojuegos usan las llamadas "pistas auditivas" para ayudar al jugador a entender lo que está ocurriendo en el juego. El videojuego *Overwatch* reproduce una frase de audio específica para cada personaje cuando estos ejecutan su poderoso movimiento final. Sin embargo, si no se puede escuchar esta pista, el videojuego se vuelve injusto.

Una solución es acompañar esta pista auditiva con una visual. Por ejemplo, el videojuego *Fortnite* contiene gran cantidad de pistas auditivas, los cofres que contienen armas y munición emiten un sonido agudo, los pasos de los enemigos suenan más que los de los aliados, y, dependiendo de si el enemigo se encuentra por encima o por debajo de ti, incluso suenan con eco, como si vinieran de un sótano. Todo esto, a un jugador con un problema auditivo le pone en desventaja, sin embargo, lo que empezó siendo una ayuda para el juego en dispositivos móviles, se traspasó al juego en consolas y PC. *Fortnite* te permite cambiar estas pistas auditivas por pistas visuales, en este caso, los sonidos del juego se mostrarán como iconos alrededor de la mira que indican la posición y la distancia de la fuente del sonido.

Fig. 7 Indicadores Visuales del sonido en Fortnite[F7]

Prácticas que evitar.

Ciertas prácticas de diseño deben ser evitadas si se desea crear un juego accesible para personas con dificultades auditivas.

- Se deben evitar los puzles o segmentos de juego cuya solución sea exclusivamente sonora.
- Los subtítulos deben cubrir, como mínimo, todo el dialogo del juego, independientemente si se reproduce en una escena o como conversaciones en segundo plano.
- Se debe evitar transmitir información de forma exclusivamente sonora, especialmente si esta información es crítica para el entorno de juego. Como ataques de enemigos o posiciones de objetos obligatorios.
- Se debe evitar que el jugador solo pueda acceder a los subtítulos una vez iniciado el juego, ya que es posible que pierda la información inicial. Para ello, el juego debe incluir los subtítulos por defecto, dejar activarlos durante la escena, o desde el menú inicial.

Visual

Manejo del tamaño y color.

Aquellas personas que sufran de cualquier tipo de daltonismo tendrán problemas a la hora de distinguir elementos del juego debido a que serán incapaces de diferenciar los colores en los que estos se encuentran.

La mejor solución que se puede implementar en un videojuego para ayudar a las personas con una diversidad funcional auditiva es diseñar la estética del juego teniendo en cuenta las dificultades que estas personas tienen.

Para ello, se pueden usar distintas soluciones al diseño:

- Los elementos críticos del juego han de tener un esquema de contraste distinguible para todos los tipos de daltonismo. El azul y el naranja son distinguibles para aquellas personas con Deuteranopia, Protanopia y Tritanopia.

Fig. 8 Modo por defecto de Overwatch (Centro) comparado con distintos filtros.[F8]

- También se pueden usar colores brillantes y oscuros ya que, aunque el jugador quizá no distinga el color, podrá distinguir el cambio en la luminosidad. Aunque la primera solución es preferible.
- La mejor solución es evitar el uso exclusivo de colores. Añadiendo modificaciones fácilmente identificables a aquellos elementos del juego que lo necesiten. Ya sea mediante animaciones, estampados, patrones, o elementos externos, como cuernos, cascos, etc.
- Es importante no sobrecargar el HUD, o visor de juego. La mejor forma de transmitir información visual en este, suelen ser iconos de tamaño adecuado.
- Si el juego tiene elementos cuyo color varía (por ejemplo, los colores de un equipo en juego competitivo), dar la opción de que estos se mantengan estáticos.
- Si hay elementos que siempre se muestran en pantalla con un color estático, dar la opción de que el usuario pueda elegir ese color.
- A la hora de implementar texto, es importante que el tamaño de este sea ajustable, facilitando la lectura a las personas con baja visibilidad.
- El texto del juego, aunque se muestre en una fuente propia del juego (o la propia letra de algún personaje), debe poder siempre ser leído en una tipografía neutra, recomendablemente una Sans Serif, con al menos 46 pixeles de tamaño vertical.
- Siempre que sea posible, implementar una opción que reduzca la información del entorno. Por mucho que este ayude al juego, puede obstaculizar la lectura de este. Especialmente los juegos arcade, pueden implementar modos en los que los fondos desaparezcan. *Street Fighter* 4, incluye esta opción.
- Separar, si es posible, todos los tipos de sonido (ambiental, diálogos, SFX) en el control del volumen del menú de opciones. De tal forma que el jugador pueda elegir que sonidos suenan más que otros.
- Hacer el juego compatible con lectores de pantalla (aplicaciones que describen por audio aquello que se muestra en pantalla.)

Pistas auditivas

Es importante el uso de pistas auditivas (en conjunción con las visuales) que informen de los cambios en el entorno de juego. Los ataques de los enemigos, así como las ayudas de los aliados, deben ir acompañadas de sonido fácilmente identificable, que ayude al jugador a reaccionar. *Overwatch* hace esto de manera sobresaliente. Cuando se usa la habilidad definitiva de un personaje, este la anuncia con una frase única a dicho personaje, además, la

frase cambia dependiendo de si el personaje es enemigo o aliado, de tal forma que el jugador sepa cómo reaccionar.

También es una buena práctica, ofrecer al jugador la posibilidad de eliminar el ruido de fondo, o subir el volumen de ciertas pistas, dado que esto ayudará a que las diferencie con mayor facilidad. *Mortal Kombat* es jugable incluso por personas con ceguera total [11], debido a que deja modificar estas opciones.

Prácticas que evitar.

Hay ciertas prácticas de diseño que han de ser evitadas para poder facilitar la accesibilidad

- Tener elementos que solo se muestren visualmente, especialmente si se tratan de elementos a los que se espera que el jugador reaccione con rapidez.
- Diseñar elementos que se comportan de forma diferente con una estética similar, ya que estos no serán distinguibles.
- Evitar el uso de una paleta de colores oscura y con poco contraste en el diseño, en el que los elementos con los que se pueda interactuar no sobresalgan
- Evitar elementos con color variable (por ejemplo, los colores de dos equipos de una partida)
- Tener un solo esquema visual, es mejor si el jugador puede elegir ciertos colores del juego.

iii. Cognitivas

La diversidad funcional cognitiva engloba varios tipos de enfermedades, desde epilepsia a dislexia pasando por todo tipo de problemas de aprendizaje. En este apartado se darán soluciones comunes a todas, pero se especificarán si alguna solución es particular para algún tipo de enfermedad.

Manejo de la información.

La diversidad funcional cognitiva se caracteriza por un problema en el procesamiento de la información, por ello, es especialmente importante prestar atención a la cantidad de información que mostramos en pantalla y cómo la mostramos.

- La información crítica debería expresarse siempre en términos sencillos. Esto se refiere, por ejemplo, al objetivo de una misión que se desbloquee al hablar con el personaje de un videojuego. El objetivo final debería ser claro.
- Es recomendable que la información acerca del objetivo se muestre de forma constante en pantalla mediante una versión resumida y comprensible del objetivo actual.
- Si no se muestra de forma constante, la información ha de poder consultarse en cualquier momento desde un menú, normalmente el de pausa.
- También es recomendable que el jugador pueda acceder a una versión más detallada de ese objetivo si así lo requiere, dado que es posible que con la versión simplificada no quede claro.
- Aunque es una práctica común, es importante un modo de pausa que detenga completamente el juego, dando tiempo al jugador a poder evaluar la situación actual

del juego y cómo reaccionar a ella. Es aún mejor si esta opción está separada de un menú, pudiendo así ver el estado actual del juego mejor.

- También son útiles aquellas herramientas dentro del juego que permitan ver los objetos con los que el jugador puede interactuar (los llamados modos detective) o que ofrecen versiones simplificadas de la información de juego, como una vista estratégica o aérea.

Fig. 9 Visión Hexagonal de Civilization V. Que muestra la información de manera simple[F9]

- En los juegos que disponen de una cantidad de texto escrito, es importante mostrar este texto en una letra clara, como ya hemos dicho, una Sans Serif de al menos 46 pixeles de tamaño es lo recomendable.
- Si el juego dispone de textos escritos en paredes o con caligrafías manuales de algún personaje, es recomendable que se pueda ver esta información como un texto normal, con la fuente ya citada.
- También es recomendable evitar que los textos avancen de forma automática, ya que limitan el tiempo del que dispone el jugador para leer el texto.
- Una práctica poco común, pero recomendable es la inclusión de un registro de audio, en el que se guarden los diálogos que han sido escuchados por el jugador, de tal forma que, al acabar una conversación, pueda releerla tranquilamente si tiene dudas.
- También es importante la inclusión de tutoriales y que estos sean accesibles de nuevo en cualquier momento para que el jugador pueda recordar ciertos controles en cualquier momento.
- Los modos de práctica también son útiles ya que permiten al jugador ensayar habilidades o situaciones en un entorno sin estrés.
- La inclusión de los denominados "*save states*" o estados de guardado es también útil, ya que permiten al jugador guardar la partida en cualquier momento (por ejemplo, antes de un enfrentamiento), y volver rápido a ese punto para probar distintas estrategias. Es mejor si estos estados incluyen una imagen de la pantalla en ese momento, ya que facilita al jugador la distinción de los guardados.

- Es útil una guía para llegar al objetivo, esta puede mostrarse de distintas formas. Desde una flecha flotante que apunte en la dirección a seguir, a un GPS o una luz brillante en el cielo. Lo importante es que el jugador siempre pueda saber a dónde debe ir para continuar el camino crítico del juego.

Prácticas que evitar.

A la hora de diseñar un juego accesible para las personas con diversidad funcional cognitiva, existen ciertas prácticas que es necesario evitar.

- Elaborar misiones complejas o poco concisas. Especialmente si el jugador no puede acceder a los objetivos de una misión. Si una misión tiene varios objetivos, lo mejor es, siempre que sea posible, separarla en varias misiones. Si no es posible, es recomendable que los objetivos vayan uno tras otro, es decir, que el jugador tenga un objetivo concreto al que acudir en cada momento.
    - Si el jugador puede elegir entre varios objetivos en un momento dado, es importante resaltar estos objetivos y cuales son. Si el juego dispone de algún sistema de guía, siempre debe apuntar al objetivo que haya que cumplir en primer lugar, o al objetivo más cercano.
    - Si existen varias formas distintas de cumplir un objetivo principal, el juego debe anotarlas todas, conforme el jugador las descubra. Además, debe ir registrando el progreso de cada objetivo.
- Usar tipografías propias de manera obligatoria. Las fuentes propias de un juego son una práctica relativamente común, pero normalmente no son muy legibles. Si el juego dispone de una tipografía propia, es importante que el jugador pueda cambiar a una fuente secundaria más legible.
- Usar cortes de cámara muy bruscos de manera rápida o luces parpadeantes muy brillantes. Estas prácticas pueden causar ataques epilépticos en los jugadores más propensos a estos, y también pueden causar mareos por simulación.
- Usar movimientos de cámara rápidos de manera continuada, ya que causan mareos por simulación.

iv. Motoras

Las diversidades funcionales motoras suelen ser dadas por enfermedades degenerativas o genéticas. Existe una creencia errónea de que la accesibilidad de un producto frente a una diversidad motora ha de enfrentarse únicamente desde el punto de vista del hardware, y que, por lo tanto, a la hora de desarrollar un producto software no es necesario prestarle atención. Aunque la accesibilidad por parte del hardware tiene un gran impacto en las diversidades motoras, existen ciertas prácticas en el desarrollo del Software que pueden ayudar mucho.

Sistemas de controles.

A la hora de diseñar un videojuego accesible para personas con problemas motores, debemos centrar nuestros esfuerzos en el diseño del sistema de controles del juego y en cómo vamos a pedir a los jugadores que interactúen con él.

- Incluir sistemas de controles simplificados. Si tu juego dispone de un sistema muy complejo, especialmente si obliga a mantener varios botones pulsados al mismo

tiempo, es importante incluir un sistema simplificado de controles, que disponga de las mismas funciones, pero usando menos botones.

- Dejar personalizar los controles de "mantener" y "doble pulsación". Es posible que para ciertos jugadores sea incomodo o difícil mantener pulsado un botón durante mucho tiempo o pulsarlo varias veces de manera muy rápida. La solución más sencilla es dejar sustituir estos controles. Por ejemplo, si el jugador debe mantener pulsado un botón para apuntar, es bueno que pueda cambiar este control para empezar a apuntar al pulsar un botón y dejar de apuntar al volver a pulsarlo.
- Incluir un sistema de dificultad variado. Es importante que este sistema sea respetuoso con los nombres, ya que existe una tendencia a nombrar los modos de dificultad más sencillos de formas que podrían ser ofensivas. Personalizar la dificultad siempre es agradecido, ya que permite al jugador crear una propia experiencia de desafío adecuada a sus capacidades.
- Incluir un sistema de resolución de "*Quick Time Events*" (QTE). Los QTE son un tipo de minijuego muy común que consiste en hacer al jugador pulsar una serie de botones de manera rápida y constante, normalmente mientras transcurre una escena de fondo. Es importante que el jugador pueda optar por "resolver automáticamente" estas escenas, o que tengan una dificultad ajustada.
- Incluir compatibilidad con distintos tipos de controladores. Mandos de todo tipo, teclados, ratones, etc. Existen controladores con software específico para un tipo de mando que son mas accesibles para ciertos perfiles de diversidad funcional motora. Permitiendo usar varios controladores distintos, permitiremos al jugador usar aquel con el que se sienta más cómodo.

Practicas a evitar.

- Inclusión de una mecánica o QTE que fuerce al jugador a mantener un botón pulsado o pulsarlo de forma repetida en un corto periodo de tiempo.
- Imponer un sistema de controles al jugador.
- No hacer el juego compatible con distintos tipos de mandos o controles

v. Conclusiones

La accesibilidad de cualquier producto es variable, desde un nivel nulo hasta alcanzar la accesibilidad universal (producto completamente accesible), aunque conseguir satisfacer este último nivel es verdaderamente complicado. En este apartado solo se han citado las prácticas más usadas actualmente en la industria del videojuego, y aun así es muy difícil encontrar un videojuego que implemente de manera acertada la mayoría de estas prácticas.

En principio, la accesibilidad suele ser fácil de desarrollar y barata, a Ubisoft sólo le costó 8.5 hombres/día completar los subtítulos de *Assasins Creed Origins*. Además, mejora la calidad del videojuego, no solo acercándolo a un mayor número posible de jugadores, sino haciéndolo más disfrutable para aquellos que podrían jugarlo sin aprovechar dichas opciones. Principalmente se basa en darle la opción al jugador de elegir sobre los ajustes del juego, cosas como la dificultad, el esquema de color, la tipografía o el tamaño de los textos pueden ayudar enormemente a diversas personas a disfrutar de un videojuego que, si no fuera por estos ajustes, no supondría un desafío divertido sino un castigo injusto.

Hemos observado que muchas de las opciones de accesibilidad se complementan unas a otras, lo referente a lo cognitivo, lo visual y lo auditivo funciona mejor cuando se implementan medidas en todos los frentes. Aunque el área motora parece ser la más olvidada y normalmente relegada al hardware, es importante también tomar medidas que, de no ser tomadas, pueden hacer imposibles para algunas personas la superación del videojuego.

En definitiva, la accesibilidad es algo que, en mayor o en menor medida, siempre debería ser implementado en un videojuego. Mejora la calidad del mismo, no es cara de implementar y puede ser alcanzada cumpliendo unos pequeños pasos en la dirección correcta.

d. Conclusiones del estado del Arte.

Existen gran cantidad de distintos perfiles de diversidad funcional, intentar desarrollar soluciones específicas para cada uno de ellos no es viable. Sin embargo, muchos de estos perfiles comparten características, haciendo así posible desarrollar soluciones comunes a ellos.

En distintos medios, la accesibilidad está más desarrollada, el cine o la televisión cuentan con unas directrices marcadas en lo que a la accesibilidad se refiere, sin embargo, en el mundo de los videojuegos, sólo se ha empezado a desarrollar de manera reciente. Los videojuegos 3D en particular suponen un desafío mayor, al simular un entorno que dispone de iluminación, espacio, cualidades físicas, sonidos, etc. Por desgracia, videojuegos con amplias opciones de accesibilidad son escasos en la industria, y no se desarrollan las soluciones necesarias de manera habitual.

Es impensable hacer un videojuego 3D sin un motor de videojuegos que lo soporte. Algunos de ellos, como Unreal Engine, disponen de herramientas que permiten desarrollar más fácilmente la accesibilidad.

En general, en nuestra opinión la mayoría de las soluciones ya son conocidas y en muchos casos hasta han sido desarrolladas previamente. En una industria creciente, empiezan a ser más comunes, pero están aún lejos de ser la norma, y no existen demasiados recursos que ayuden a los desarrolladores a encontrar e implementar éstas soluciones. Es un hecho objetivo que la accesibilidad mejora la calidad de cualquier producto, en particular los videojuegos, por lo que siempre es algo deseable.

Toda la información recolectada durante este apartado construye una buena guía sobre qué tener en cuenta a la hora de diseñar un producto Software accesible.

4. Hipótesis de Trabajo.

En este apartado se explicará el desarrollo del trabajo realizado. Este prototipo se lleva a cabo para ver las posibilidades en la realidad de desarrollar un videojuego accesible con una herramienta popular en el mercado actual. Primero se hará un análisis del proyecto a realizar acompañado de un estudio de la herramienta a utilizar para llevar acabo dicho proyecto. También se explicará la Rúbrica creada para valorar la accesibilidad de un videojuego. Por último, se aportarán conclusiones acerca del trabajo realizado y lo aprendido durante el desarrollo.

a. Análisis, Diseño y desarrollo del prototipo accesible

El objetivo de esta sección es definir la funcionalidad que tendrá el prototipo, así como los requisitos que se han de cumplir en el desarrollo de este. Después, se realizara un estudio de la herramienta a utilizar con el objetivo de ver qué posibilidades de accesibilidad nos ofrece. También se hará una explicación de los diferentes casos de uso del prototipo, acompañada de diagramas. Por último, se explicará todo el trabajo que se ha realizado en el desarrollo del prototipo.

i. Análisis de los requisitos del prototipo.

Se va a realizar la elaboración de un prototipo de videojuego accesible con el motor Unreal Engine en su versión 4.20.3. Este apartado del documento detallará los requisitos funcionales del proyecto, así como la justificación de dichos requisitos.

A continuación, se muestra la lista de Requisitos Funcionales (RF)

- RF01. El juego debe de iniciarse en un Menú Principal.
- RF02. Ha de poderse salir del juego en cualquier momento.
- RF03. Ha de existir una pantalla explicando todos los controles del juego.
- RF04. El objetivo del juego ha de ser recoger objetos en un entorno.
- RF05. El juego acabará una vez se hayan recogido todos los objetos.
- RF06. El juego ha de poder jugarse en su totalidad solo con el ratón y sólo con el teclado.
- RF07. El juego ha de tener un esquema de colores de alto contraste.
- RF08. El juego ha de tener pistas sonoras para indicar los objetos a coleccionar cercanos.
- RF09. El juego ha de tener pistas visuales que resalten los objetos a recoger.
- RF10. El juego ha de tener pistas sonoras que indiquen cuando se ha recogido un objeto.
- RF11. El juego ha de poder pausarse en cualquier momento.
- RF12. El juego ha de tener pistas visuales que indiquen cuando se ha recogido un objeto.
- RF13. El juego ha de tener una pantalla de victoria que se muestre cuando se recojan todos los objetos.

- RF14. El juego ha de mostrar en todo momento cuántos objetos se han recogido.
- RF15. El juego ha de mostrar en todo momento cuántos objetos hay en total.
- RF16. El juego ha de mostrar en todo momento el objetivo del juego.
- RF17. El juego ha de tener una forma de localizar los objetos de forma auditiva.
- RF18. El menú principal y el de controles deben tener alguna forma de lectura de pantalla.

A continuación, se detallan las justificaciones para cada requisito:

- RF01: Este requisito se asegura de que el jugador tenga un momento para prepararse antes de entrar al juego, en lugar de ser lanzado directamente a la acción. Así, el jugador puede familiarizarse con los controles, o salir del juego si lo ha iniciado por error.
- RF02: Este requisito facilita la salida del juego, si el jugador se viera sobrepasado, o necesitará dejar de jugar de forma inmediata, ha de poder salir del juego rápidamente.
- RF03: Una pantalla que explique los controles del juego permite al jugador hacerse a este de antemano, asegurarse de que conoce las posiciones de las teclas y prepararse para iniciar el juego sabiendo como moverse por él.
- RF04: Este requisito sirve para asentar las bases del juego, lo importante de este juego es la exploración, ser capaz de familiarizarse con un entorno virtual y recorrerlo de manera cómoda. Al tener que buscar objetos en un entorno, el juego anima al jugador a moverse por este, aprender el entorno y como interactuar con él.
- RF05: Este requisito limita el juego, una vez se ha cumplido el objetivo, no existe mayor razón para jugar. Cuando se ha completado el objetivo, el juego se acaba.
- RF06: Un esquema de control sencillo facilita la accesibilidad para los perfiles cognitivos y motores. Si el juego puede jugarse con solo dos botones, es fácil aprender a usarlo.
- RF07: Un esquema de colores de alto contraste es imprescindible para la accesibilidad de perfiles visuales ya que permite distinguir las partes del juego fácilmente, pudiendo así interactuar con ellas sin problemas.
- RF08: Las pistas sonoras también ayudan en la accesibilidad de perfiles visuales, permitiendo, en este caso, localizar, objetos que quizás no pudieran verse bien.
- RF09: Las pistas visuales acercan el juego a los perfiles auditivos, pudiendo distinguir fácilmente los objetivos del resto de modelos del juego.
- RF10: Al igual que con el RF08, las pistas sonoras ayudan a saber cuándo se ha recogido un objeto.
- RF11: Poder pausar el juego en cualquier momento permite al jugador disfrutar del juego bajo sus propios términos, pudiendo parar el juego para evaluar su situación o intentar conocer mejor los controles.

- RF12: Al igual que con el RF09, las pistas visuales ayudan a saber si un objeto ha sido recogido a aquellas personas con diversidad funcional auditiva.
- RF13: Dado que el juego se acaba una vez se alcanza el objetivo de Victoria, una pantalla de victoria ayuda a comunicar al jugador que ya ha acabado, y que puede dejar de jugar. Esta pantalla se asegura de que el jugador entienda que el juego ha acabado, antes de salir él cuando sea necesario.
- RF14: Este requisito está orientado a los perfiles cognitivos, ayudando a saber en qué situación se encuentra el juego en cada momento
- RF15: Este requisito se complementa con el RF14, aumentando la cantidad de información acerca del estado del juego.
- RF16: Al igual que el RF14, este requisito se asegura de que el jugador siempre sepa que debe hacer, evitando así que sienta confusión durante el juego.
- RF17. Con un sistema auditivo de localización, cualquier jugador con un perfil visual puede localizar fácilmente el objeto. Gracias a que los objetos brillan, no afecta a los perfiles auditivos.
- RF18. Un sistema de lectura de pantalla de los menús principales ayuda al jugador a moverse fácilmente por estos.

ii. Casos de Uso.

En este apartado se detallarán los diferentes casos de uso del prototipo, explicados pantalla por pantalla.

Menú Principal

El menú principal se iniciará según se inicie el juego, cumpliendo así el requisito funcional 1. Este es el diagrama de casos de uso del menú principal:

A continuación, se explican los casos de uso del menú principal.

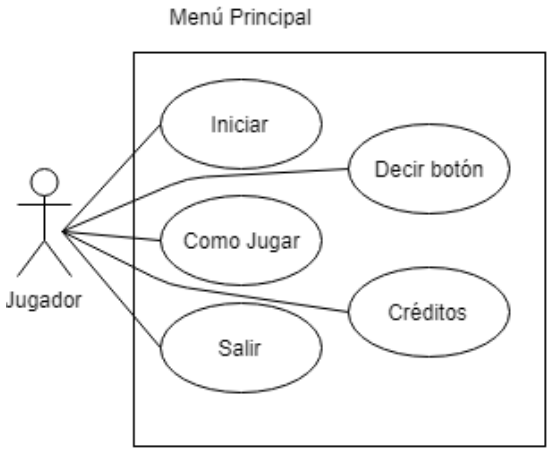

Fig. 10 Diagrama de Casos de Uso del Menú principal

CU-01	Iniciar

Dependencias	RF01 – El juego ha de tener un menú principal
Precondiciones	Ninguna
Descripción	El sistema debe iniciar el juego cuando se dé este Caso de Uso
Flujo	1. El jugador, desde el menú principal, pulsa el botón de Iniciar 2. Se carga el nivel de juego. 3. El jugador obtiene el control del juego.
Postcondiciones	Empieza el nivel de juego.

CU-02	**Como Jugar**
Dependencias	RF03 – El juego ha de tener una pantalla que explique los controles.
Precondiciones	Ninguna
Descripción	El sistema debe abrir la pantalla de Como Jugar cuando se dé este caso de uso
Flujo	1. El jugador, desde el menú principal, pulsa el botón de Como Jugar 2. Se descarga el menú principal. 3. Se carga el menú de controles.
Postcondiciones	Se carga el menú de controles.

CU-03	**Créditos**
Dependencias	Ninguna
Precondiciones	Ninguna
Descripción	El sistema debe mostrar el menú de creditos
Flujo	1. El jugador, desde el menú principal, pulsa el botón de Salir 2. Se descarga el menú principal 3. Se carga el menú de créditos.
Postcondiciones	Se muestra el menú de creditos

CU-04	**Salir**
Dependencias	RF02 – Se ha de poder salir del juego en cualquier momento.
Precondiciones	Ninguna
Descripción	El sistema debe salir del juego completamente cuando se dé este caso de uso.
Flujo	1. El jugador, desde el menú principal, pulsa el botón de Salir 2. Se cierra la ventana de juego
Postcondiciones	Se cierra la ventana de juego.

CU-05	**Decir Botón**
Dependencias	RF18. El menú principal y el de controles deben tener alguna forma de lectura de pantalla.
Precondiciones	Ninguna
Descripción	El sistema debe leer el botón que esta seleccionado
Flujo	1. El sistema detecta que botón esta seleccionado 2. El sistema reproduce un audio que lea ese botón
Postcondiciones	Ninguna

Cómo Jugar

La pantalla de Cómo Jugar es la encargada de mostrar los controles del juego, cumpliendo con el RF03.

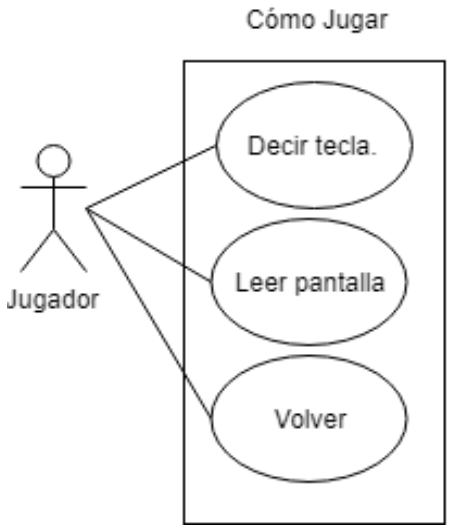

Fig. 11 Diagrama de casos de uso de Cómo Jugar

CU-06	Decir Tecla
Dependencias	Ninguna
Precondiciones	Ninguna
Descripción	El sistema debe emitir un sonido que diga la tecla a pulsar para un control.
Flujo	1. El jugador pulsa un botón que indica la tecla asignada a un control del juego 2. El sistema emite un archivo de audio en el que se diga la tecla indicada
Postcondiciones	Ninguna

CU-07	Leer Pantalla
Dependencias	RF18. El menú principal y el de controles deben tener alguna forma de lectura de pantalla.
Precondiciones	Ninguna
Descripción	El sistema debe leer la pantalla
Flujo	1. El sistema abre el menú de cómo jugar 2. El sistema lee la pantalla de cómo jugar.
Postcondiciones	Se cierra la ventana de juego.

CU-08	Volver
Dependencias	Ninguna
Precondiciones	Ninguna
Descripción	El sistema debe dejar de mostrar el menú de controles y mostrar el Menú Principal
Flujo	1. El jugador pulsa el botón de Volver

	2. Se deja de mostrar el menú de cómo jugar 3. Se muestra el Menú Principal
Postcondiciones	Se carga el Menú Principal

Nivel

El Nivel es el propio juego, en el que el jugador toma el control de una Nave y ha de buscar rocas.

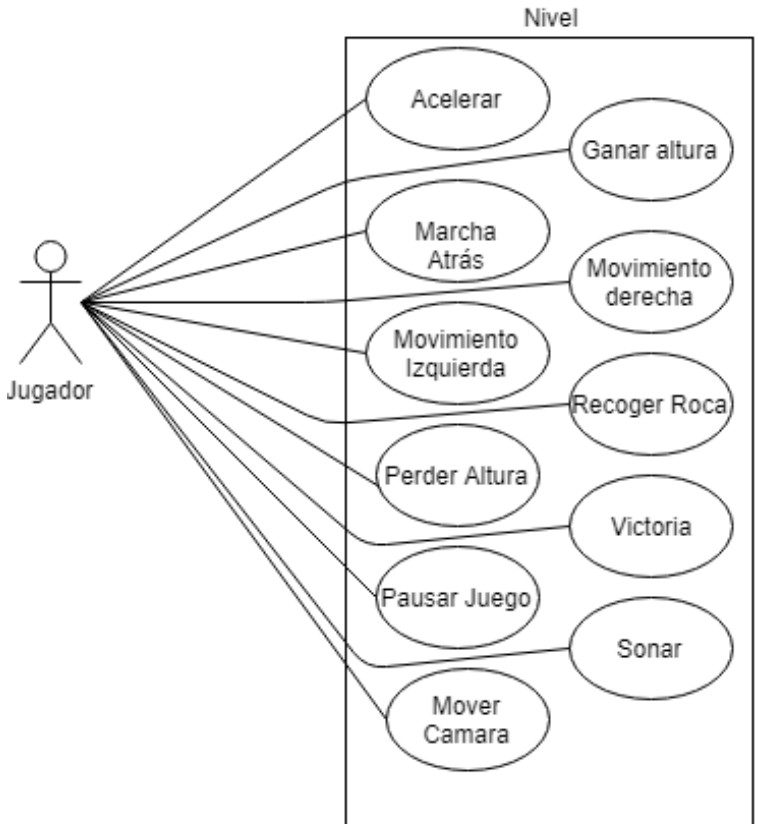

Fig. 12 Diagrama de Casos de Uso del Nivel

CU-09	Acelerar
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	El peón del jugador se moverá en la dirección que mira la cámara.
Flujo	1. El jugador usa el control de acelerar 2. El juego calcula el vector hacia el cual está apuntando la cámara. 3. Si no hay colisión bloqueante con un objeto en ese vector 3.1. El peón se mueve en esa dirección a una velocidad constante. 4. Si hay colisión bloqueante con un objeto en ese vector. 4.1. El peón rebota cambiando de dirección. 4.2. El sistema emite un sonido indicando que el peón ha rebotado
Postcondiciones	Cambia la posición del peón en el mapa.

CU-10	Ganar Altura
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	El peón del jugador ha de aumentar en altura respecto al mapa.
Flujo	1. El jugador usa el control de Ganar Altura. 2. Si no hay colisión bloqueante con un objeto en el vector Z del peón 2.1. El peón se mueve en ese vector a una velocidad constante. 3. Si hay colisión bloqueante con un objeto 3.1. El peón se queda en posición.
Postcondiciones	Si se cumple la condición 2, el peón gana altura en el mapa.

CU-11	Marcha atrás
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	El peón del jugador ha de moverse hacia la cámara.
Flujo	1. El jugador usa el control de Marcha Atrás 2. El juego calcula el vector hacia el cual apunta la cámara. 3. Si no hay colisión bloqueante en la inversa de ese vector. 3.1. El peón avanza en el vector inverso. 4. Si hay colisión bloqueante en el vector inverso, 4.1. el peón se queda en posición
Postcondiciones	Si se cumple la condición 2, el Peón retrocede.

CU-12	Movimiento a la derecha
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	El peón del jugador se mueve de forma relativa al peón hacia la derecha.
Flujo	1. El jugador usa el control de Movimiento a la Derecha. 2. El juego calcula el vector que apunta hacia la derecha del peón. 3. Si no hay colisión bloqueante con un objeto en dicho vector 3.1. El peón avanza en ese vector. 4. Si hay colisión bloqueante con un objeto en ese vector 4.1. El peón se mantiene en posición
Postcondiciones	Si se cumple la condición 3, el peón cambia de posición.

CU-13	Movimiento a la Izquierda
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna

Descripción	El peón del jugador se mueve de forma relativa hacia la Izquierda.
Flujo	1. El jugador usa el control de Movimiento a la Izquierda 2. El juego calcula el vector que apunta hacia la derecha del peón 3. Si no hay colisión bloqueante con un objeto a la inversa de ese vector 3.1. El peón del jugador avanza en el vector inverso. 4. Si hay colisión bloqueante con un objeto a la inversa de ese vector 4.1. El peón del jugador se mantiene estático.
Postcondiciones	Si se cumple la condición 3, el peón cambia de posición.

CU-14	Recoger Roca
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	La colisión del peón del jugador debe estar colapsando con la colisión de una Roca
Descripción	El peón del jugador ha de recoger la roca cuando la atraviese.
Flujo	1. El sistema detecta que la colisión del peón del jugador está colapsando con la colisión de una Roca. 2. El contador de rocas recogidas aumenta 3. La roca emite una pista auditiva única que indique que se ha recogido. 4. La roca emite una pista visual única que indique que se ha recogido. 5. La roca desaparece del mapa.
Postcondiciones	La roca desaparece del mapa. El contador de rocas recogidas aumenta en uno.

CU-15	Perder Altura
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	El peón del jugador pierde altura en el mapa
Flujo	1. El jugador usa el control de Perder Altura 2. El sistema calcula la inversa al vector Z del peón del jugador. 3. Si no hay colisión bloqueante con un objeto en ese vector 3.1 El peón del jugador avanza en dicho vector. 4. Si hay colisión bloqueante con un objeto en ese vector 4.1 El peón del jugador se mantiene en posición.
Postcondiciones	Si se cumple la condición 3, cambia la posición del peón en el mapa

CU-16	Pausar Juego
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	El juego se pausa y se muestra un menú de pausa.

Flujo	1. El jugador usa el control de Pausar Juego. 2. El juego se pausa en su totalidad. 3. Se habilita el control del cursor. 4. Se muestra el Menú de Pausa
Postcondiciones	Se carga el Menú de Pausa.

CU-17	**Victoria**
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	Se muestra la pantalla de victoria.
Flujo	1. El sistema detecta que se cumple la condición de Victoria. 2. El jugador pierde el control del peón. 3. Se descarga el Nivel de juego. 4. Se carga el menú de victoria. 5. Se emite un sonido que indique que se ha alcanzado la victoria.
Postcondiciones	Se carga el Menú de Victoria.

CU-18	**Sonar**
Dependencias	RF17. El juego ha de tener una forma de localizar los objetos de forma auditiva.
Precondiciones	Ninguna
Descripción	El sistema reproduce un sonido que indique la posición de la roca más cercana
Flujo	1. El sistema calcula la distancia a todos los objetos 2. El sistema escoge el objeto a recoger que sea más cercano 3. El sistema reproduce un sonido que localice espacialmente dicho objeto
Postcondiciones	Ninguna

CU-19	**Mover cámara**
Dependencias	RF06 – El juego ha de poder jugarse solo con el ratón o solo con el teclado
Precondiciones	Ninguna
Descripción	Se mueve la cámara
Flujo	1. El sistema detecta en qué dirección se ha de mover la cámara, ya sea por teclado o por ratón. 2. El sistema mueve la cámara
Postcondiciones	Se cambia la posición de la cámara.

Victoria

La pantalla de Victoria es la encargada de informar al jugador de que el juego ha acabado.

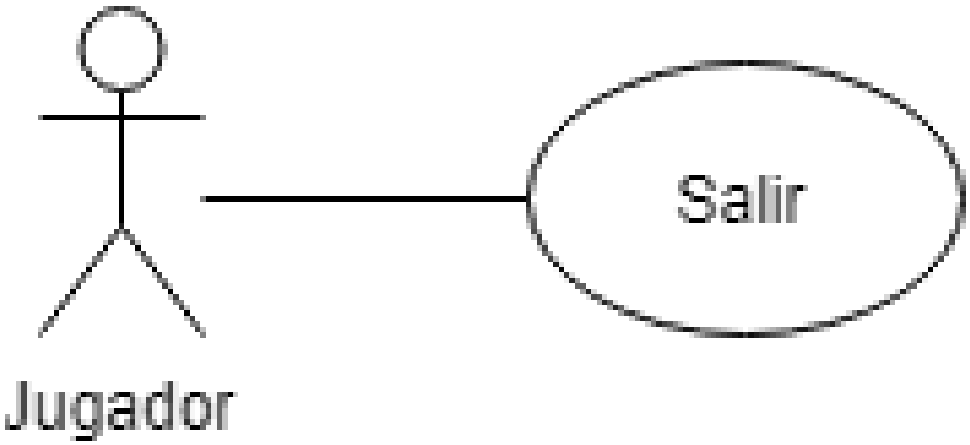

Fig. 13 Diagrama de Casos de Uso de Victoria

CU-20	Salir
Dependencias	RF02 – Se ha de poder salir del juego en cualquier momento.
Precondiciones	Ninguna
Descripción	El sistema debe salir del juego completamente cuando se dé este caso de uso.
Flujo	1. El jugador, desde el menú de Victoria, pulsa el botón de salir del juego 2. Se cierra la ventana de juego
Postcondiciones	Se cierra la ventana de juego.

Menú de Pausa

El Menú de Pausa es el que se muestra cuando, desde el nivel, el jugador pausa el juego, esto esta así hecho para cumplir el RF11.

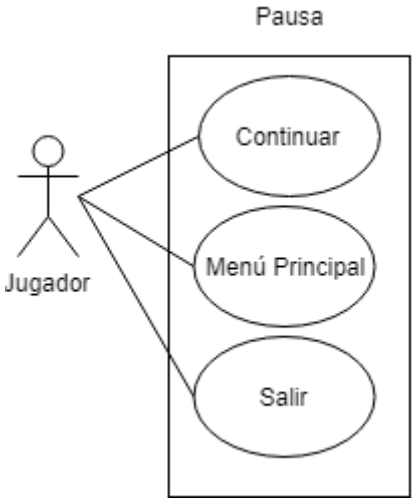

Fig. 14 Casos de Uso del menú de Pausa

CU-21	Continuar
Dependencias	RF11– Se ha de poder pausar el juego en cualquier momento.
Precondiciones	Ninguna
Descripción	El juego se reanuda
Flujo	1. El jugador pulsa el botón de Continuar 2. Se elimina la interfaz de pausa de la pantalla. 3. Se le devuelve el control del peón al jugador. 4. Se devuelve el movimiento al juego.
Postcondiciones	El jugador tiene el control del peón.

CU-22	Menú Principal
Dependencias	RF11– Se ha de poder pausar el juego en cualquier momento.
Precondiciones	Ninguna
Descripción	El sistema debe salir del juego completamente cuando se dé este caso de uso.
Flujo	1. El jugador pulsa el botón de Menú principal 2. Se elimina de la pantalla el menú de Pausa. 3. Se carga el menú principal.
Postcondiciones	Se cierra la ventana de juego.

CU-23	Salir
Dependencias	RF02 – Se ha de poder salir del juego en cualquier momento. RF11– Se ha de poder pausar el juego en cualquier momento.
Precondiciones	Ninguna
Descripción	El sistema debe salir del juego completamente cuando se dé este caso de uso.
Flujo	1. El jugador, desde el menú de pausa, pulsa el botón de Salir 2. Se cierra la ventana de juego
Postcondiciones	Se cierra la ventana de juego.

Créditos

El menú de créditos, disponible desde el menú principal, muestra los créditos del juego.

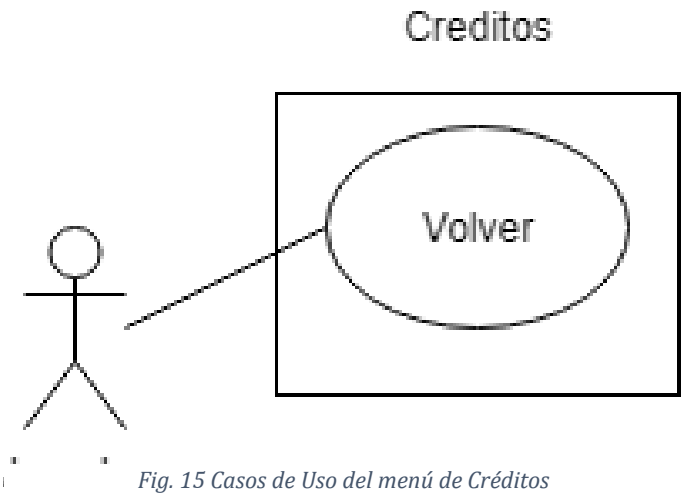

Fig. 15 Casos de Uso del menú de Créditos

CU-23	Volver
Dependencias	Ninguna
Precondiciones	Ninguna
Descripción	El sistema debe dejar de mostrar el menú de creditos y mostrar el menú principal
Flujo	1. El jugador, desde el menú de pausa, pulsa el botón de Salir 2. Se cierra la ventana de juego
Postcondiciones	Se cierra la ventana de juego.

iii. Análisis de la herramienta: Unreal Engine.

En este apartado, nos centraremos en Unreal Engine, concretamente, en su versión actual, que es la cuarta. Detallaremos los pasos necesarios para instalarla, así como un pequeño manual de sus partes principales y como se usan. Por último, valoraremos qué ventajas y desventajas tiene dicha herramienta a la hora de trabajar en la accesibilidad.

Instalación de la herramienta.

Unreal Engine 4 está desarrollado por *Epic Games*. Instalar la herramienta es bastante sencillo, y existen varias formas de hacerlo. Lo primero que debemos hacer es acceder al sitio web de *Epic Games*: https://www.epicgames.com/store/es-ES/. Ahí veremos que la página principal es una tienda de videojuegos, ya que *Epic Games* también es desarrolladora.

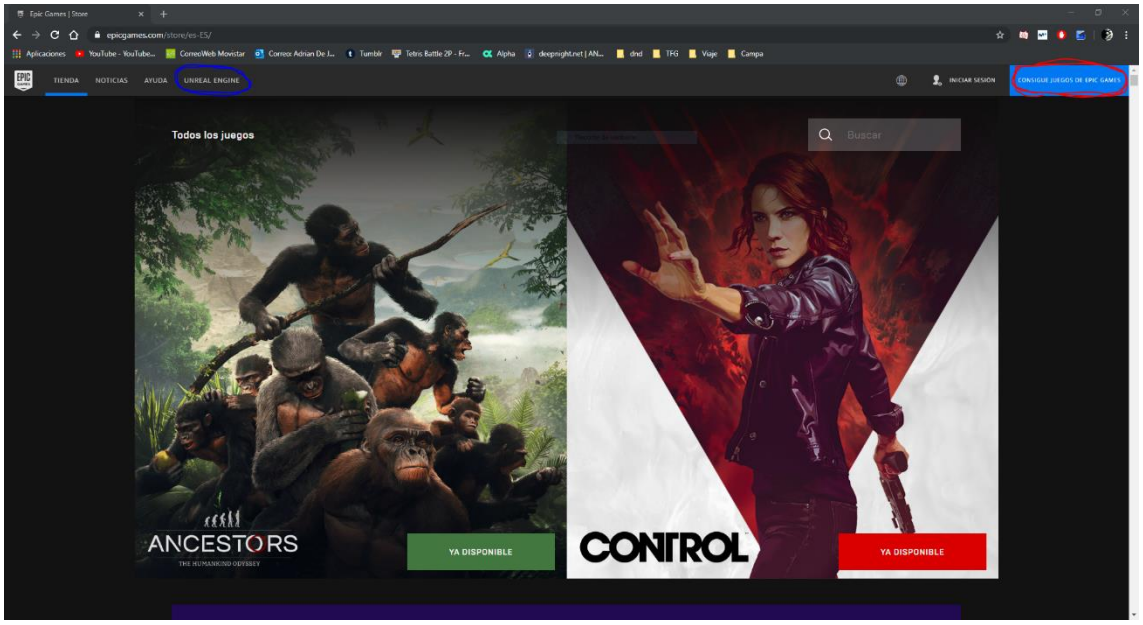

Fig. 16 Sitio Oficial de Epic Games

Para usar Unreal debemos instalarnos el programa de *Epic Games Launcher.*

Una vez en la página podemos seguir dos vínculos para hacer esto. Si clicamos en "Consigue juegos de Epic Games" (Resaltado en rojo en la figura 16), se nos descargará un instalador que nos instalará el programa necesario. Si clicamos en Unreal Engine (Resaltado en Azul en la figura 16), nos lleva a otra página, mostrada en la figura 17. En esa página, si seguimos el botón "*Download*" que se ubica arriba a la derecha, nos preguntará si queremos

descargarnos una versión para desarrolladores (gratuita) o para empresas (beta gratuita), en nuestro caso, nos descargaremos la de desarrolladores.

Fig. 17 Sitio Web de Unreal Engine

Tras haber instalado el programa, nos encontraremos en la página de Inicio de este. En el programa, se pueden apreciar distintas secciones. Inicio, tienda, biblioteca y amigos, están dedicadas a la tienda de videojuegos de Epic Games, así como a jugar online con distintas personas o ejecutar nuestros juegos. Nosotros nos dirigimos a la sección de Unreal Engine, resaltada en rojo en la figura 18

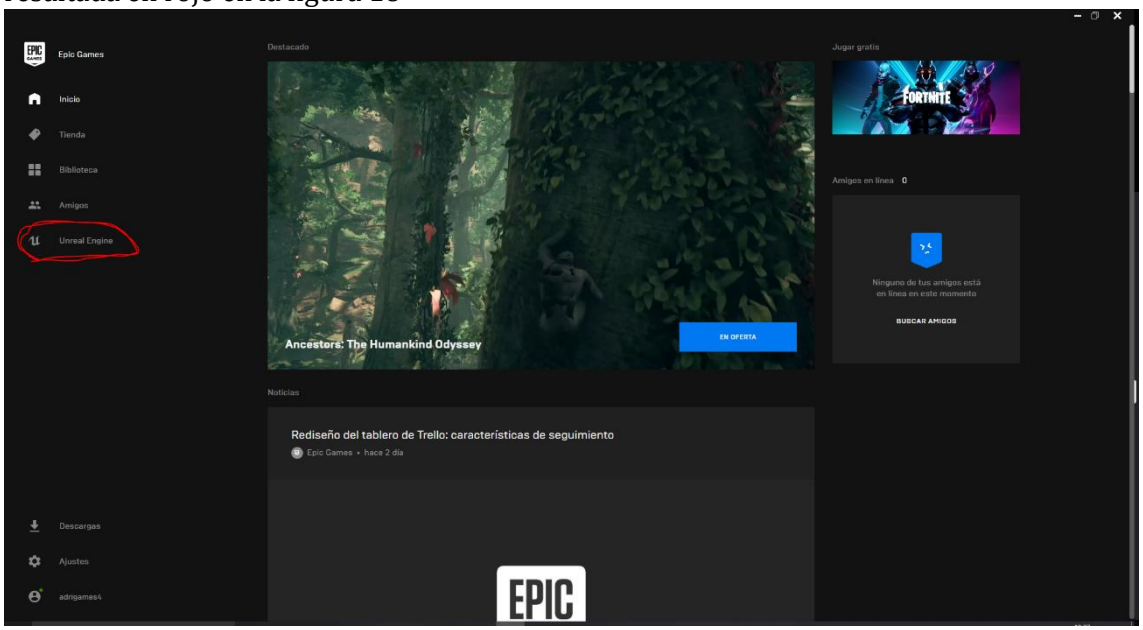

Fig. 18 Aplicación de Epic Games

Una vez hemos pulsado en Unreal Engine, veremos una diferente sección de la tienda, mostrada en la figura 19. En la parte superior podemos observar 4 pestañas. "Unreal Engine" sirve como un apartado de noticias acerca del motor, en el apartado "más información" se nos mostrarán tutoriales y recursos que ofrece *Epic Games* para

familiarizarnos con el motor. La pestaña “Bazar” sirve de tienda de recursos, en ella, podemos comprar o descargar de forma gratuita distintos recursos y extensiones de la aplicación creadas por la comunidad.

La última pestaña, biblioteca, es desde la que podremos descargarnos las distintas versiones de Unreal Engine (pulsando en el símbolo +). También se muestran las versiones que tenemos instaladas, y los proyectos que hemos creado en cada una de las versiones.

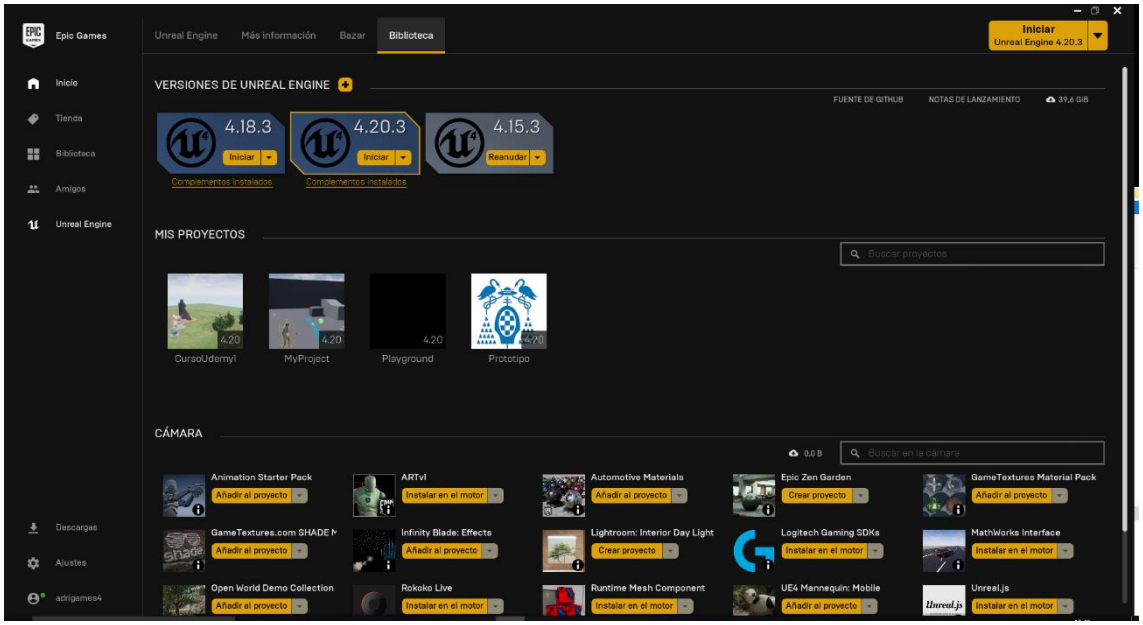

Fig. 19 Sección de Unreal Engine

Para iniciar un proyecto, solo debemos pulsar el botón iniciar, y ahí podemos seleccionar distintas plantillas con las que empezar nuestro proyecto.

Explicación de la herramienta.

Unreal Engine 4 dispone de dos formas de trabajar en un proyecto, mediante código C++, para el cual se necesita tener instalado Visual Studio 2017, y mediante Blueprints, que es un sistema propio de Unreal que abstrae los funcionamientos internos del motor, para poder trabajar mediante un sistema de codificación visual. Los Blueprints, aunque puedan sonar sencillos, son una herramienta muy avanzada, pensada para que todo tipo de profesionales puedan trabajar en el proyecto. Es posible combinar los dos tipos de programaciones en un solo proyecto, trabajando estas de manera conjunta.

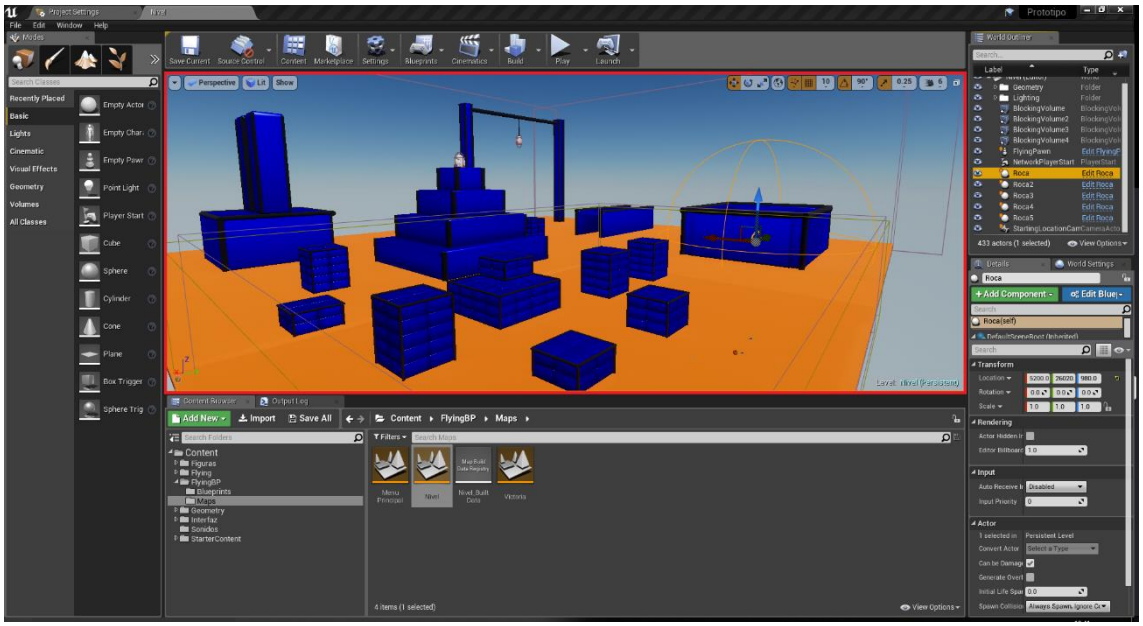

Fig. 20 Unreal Editor

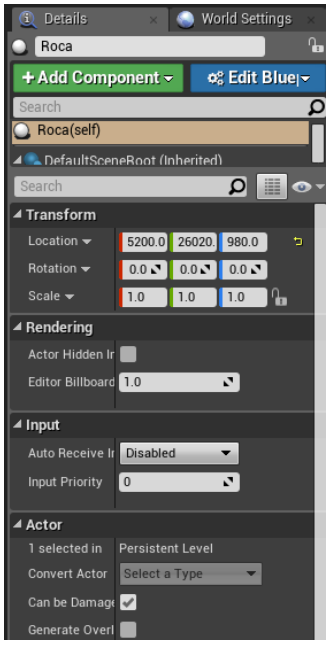

Fig. 21Panel Details y World Settings

Cuando creamos un nuevo proyecto en Unreal Engine, entramos al Unreal Editor, mostrado en la figura 20.

El Unreal Editor es la herramienta que ofrece Unreal Engine para crear cualquier videojuego. Cuando lo abrimos, podemos apreciar varios paneles distintos. El panel principal para trabajar es el *Viewport*, remarcado en rojo en la figura 20. El *Viewport* sirve a modo de "cámara" para editar el nivel y sus propiedades físicas. Desde él, podemos seleccionar cualquier objeto con un solo clic, y cambiar sus propiedades de tamaño, posición y rotación. Desde el *Viewport* también podemos movernos libremente por el nivel, comprobar las sombras que ya estén compiladas, y ver las colisiones de los objetos instanciados en el nivel.

El panel *Details*, mostrado en la figura 21, funciona como un panel de detalles, permitiéndonos ver y modificar rápidamente las propiedades de cualquier objeto que tengamos seleccionado.

El panel *World Settings*, mostrado en la figura 21, es un panel de detalles del nivel, en el podemos cambiar las configuraciones globales del nivel, asignarle música, cambiar la iluminación, etc.

El panel de *World Outliner* es un panel que permite ver todos los objetos instanciados en el nivel, desde el, se pueden organizar los objetos por carpetas, comprobar jerarquías, y hacer selecciones múltiples de objetos para realizar cambios masivos en sus propiedades. Este panel permite acceder, mediante una búsqueda, a cualquier objeto instanciado de forma rápida.

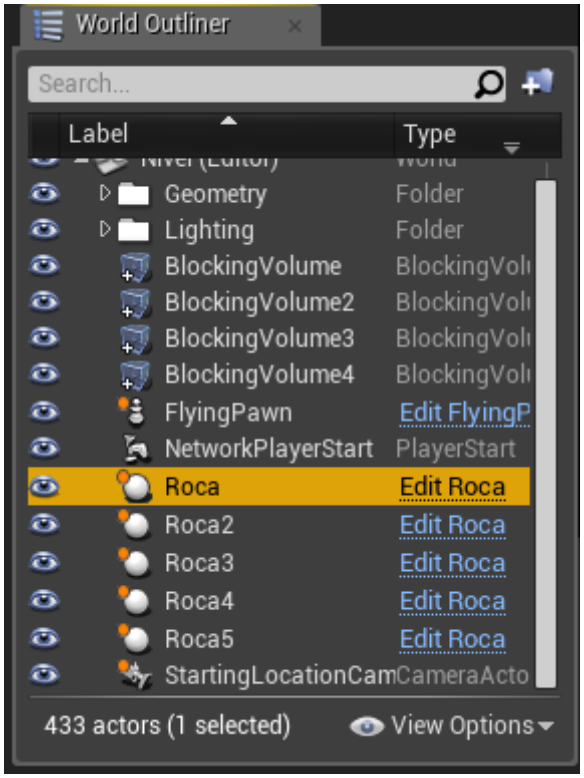

Fig. 22 Panel World Outliner

El *Content Browser* es el sistema de archivos del proyecto, desde el podemos acceder a las diferentes clases creadas para el proyecto. Todos los archivos que se vayan a usar en el proyecto deben ser importados desde este panel, o creados desde él, todos los modelos 3D, los archivos de audio, niveles, imágenes, modos de juego, interfaces, etc.

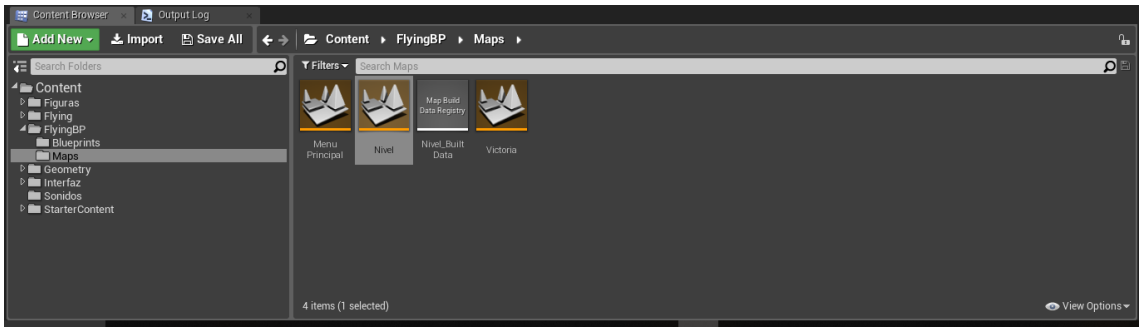

Fig. 23 Panel Content Browser

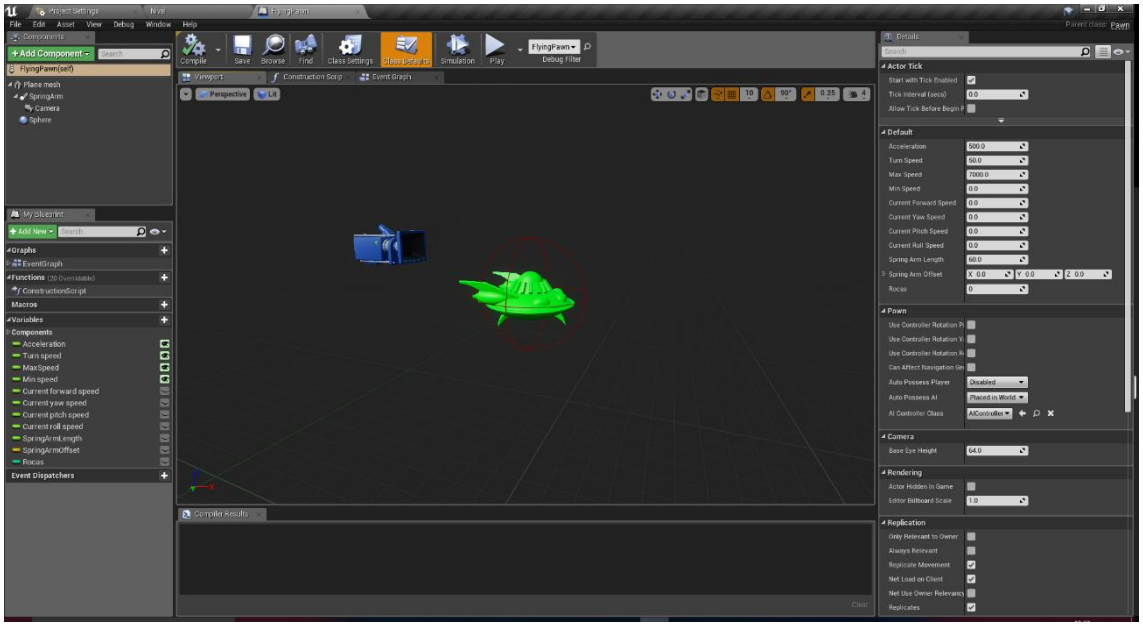

Fig. 24 Editor de Objetos

Los objetos con los que puede interactuar el usuario se llaman Actores, y si el jugador puede tomar control del objeto se le llama Peón. El resto de los objetos son simplemente objetos. Todas las clases de objetos pueden ser editadas usando el editor de objetos, combinando en un solo objeto un modelo 3D, con una colisión, audio, cámara, etc. También se le pueden fijar unos atributos por defecto al objeto, que se pondrán siempre que este sea instanciado. Por último, desde la pestaña *Event Graph* se puede crear, mediante blueprints y código C++, una funcionalidad para dicho objeto.

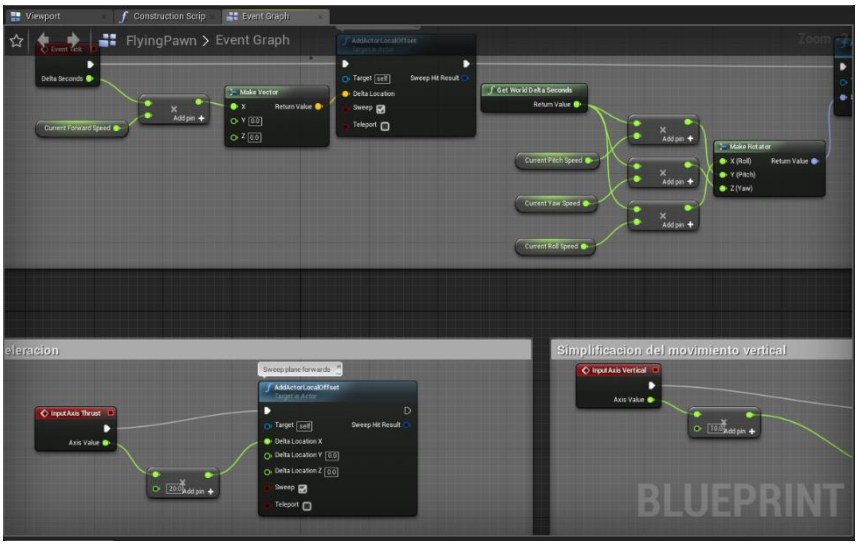

Fig. 25 Event Graph

Existen diversos editores dentro de *Unreal Editor* destinados a hacer las labores de diseño más fácil. Editores de audio, de texturas, de creación de widgets, etc. Todos estos editores son usados para crear o modificar archivos que se encuentren en el *Content Browser*. La mayoría de estos archivos disponen de un *Event Graph* destinado a darle funcionalidad.

En el *EventGraph* o sistema de eventos o planos, se pueden crear nodos. Cada nodo representa una funcion, los nodos tienen entradas (atributos a la izquierda del nodo) y salidas (atributos a la derecha del nodo). La clase o tipo de cada atributo estan marcadas con colores para que se distingan a simple vista los tipos. Para conectar un atributo o un nodo a otro, basta con arrastrar desde el tipo deseado al tipo de destino. Si estos tipos coinciden, se asignara inmediatamente. Si no coinciden, Unreal intentara hacer una conversion del tipo de origen al tipo de destino (por ejemplo, de Int a Float) añadiendo un nodo de conversion intermedio. Si dicha conversion no fuera posible, Unreal prohibe la vinculación de esos atributos.

Por último, si en la parte superior de la pantalla pulsamos en *Edit -> Project Settings* se nos abrirá la configuracion del proyecto. Aquí podemos modificar las directivas de compilación, los metadatos del proyecto, las entradas que detecta el proyecto, y toda la información relacionada con este.

Posibilidades de accesibilidad de la herramienta. Ventajas y desafíos.

Unreal Engine 4 es una herramienta principalmente pensada para diseñadores de software. Está pensada para un trabajo en equipo en el que se involucran diseñadores 3D, desarrolladores, diseñadores de videojuegos, ingenieros de sonido, etc.

Cabe destacar, que siendo uno de los principales motores del mercado, sus opciones por defecto para la accesibilidad son bastante pobres. Sólo se puede encontrar documentación dentro del programa para dos opciones de accesibilidad. La primera, los subtítulos, cualquier sonido es capaz de generar subtítulos, y que estos se añadan a la pantalla de

manera automática si así lo requiere el juego. La segunda, el juego permite tener dos modos, uno normal y otro de alto contraste, dirigido a la accesibilidad visual.

Esto presenta amplios inconvenientes, si bien estas herramientas son útiles, no son suficientes para ser dedicadas de manera específica a la accesibilidad. Por suerte, Unreal permite una gran adaptación de su entorno, siendo casi todo modificable mediante su sistema de Blueprints, del que ya se habló en el apartado anterior.

Gracias a que Unreal está dirigido a diseñadores, es sencillo crear opciones de accesibilidad, si bien se necesita un equipo detrás. Unreal permite la importación y modificación en el programa de texturas, imágenes, sonidos, estructuras 3D, etc. Prácticamente, cualquier tipo de archivo que pueda ser usado en un videojuego puede ser importado, modificado, combinado con otros archivos, e implementado en un flujo de trabajo en el que se complemente con distintas partes del proyecto. Esto es muy útil a la hora de trabajar en la accesibilidad, sin embargo, requiere una gran carga de trabajo. Todo es modificable, pero no siempre de manera sencilla.

Por ello, la comunidad ha desarrollado extensiones que facilitan ciertos trabajos, por ejemplo, algo tan simple como que el Usuario pueda cambiar el esquema de control, no es posible por defecto en Unreal, pero existen extensiones dedicadas a ello. Lo mismo ocurre con algunas de las soluciones de accesibilidad más comunes, como selectores de color para que el usuario personalice ciertos colores del juego, o el uso del *raycasting* [13] para la propagación de sonidos.

En conclusión, aunque gracias a los Blueprints y la programación en C++, todo es modificable, Unreal es un motor delicado que permite trabajar en accesibilidad, pero no siempre dispone de las herramientas para hacerlo de forma sencilla, por lo que se vuelve necesario un trabajo en equipo multidisciplinario. Ingenieros de Sonido, Diseñadores 3D, diseñadores de UI, desarrolladores, etc. Juntando conocimientos de todas estas disciplinas se pueden construir juegos accesibles, ya que el motor dispone de las funcionalidades para ello, pero no es una tarea que esté simplificada, a diferencia de otras en el motor.

iv. Desarrollo del prototipo.

Este apartado de la memoria explicará paso a paso las acciones realizadas para crear el Prototipo, justificando decisiones de desarrollo y mostrando cómo realizar ciertas funcionalidades en el motor Unreal Engine.

a. Creación del Proyecto.

El primer paso del desarrollo es crear un Proyecto, para eso, como ya se indicó en el apartado de la instalación del motor, es necesario ir a la pestaña de Unreal Engine e iniciar el motor en la versión en la que queremos desarrollar. En este caso, se ha elegido la 4.20.3 ya que es una de las últimas versiones, pero ya es estable.

Tras pulsar el botón iniciar nos sale la ventana que se muestra en la figura 26. Elegimos que el proyecto se cree por defecto con Blueprints, ya que facilita las tareas de diseño.

Aunque Unreal permite trabajar con C++, es un motor delicado, que incorpora muchos sistemas. Trabajar con C++ es útil porque permite una mayor optimización del código, sin embargo, hasta las tareas más sencillas que facilita Unreal con el sistema de Blueprints, se pueden volver muy arduas y complejas en C++, por eso sólo es recomendable para tareas de alto rendimiento en las que queramos eficiencia.

Dentro de los Blueprints, elegimos que queremos crear un proyecto de vuelo, ya que es el que más se acerca a nuestro videojuego, y eso nos permitirá que Unreal nos facilite ciertas labores de movimiento. También indicamos que el proyecto estará por el momento pensado como una aplicación de escritorio o de consola. A la hora de la verdad sólo podremos desarrollar para consolas si disponemos del SDK adecuado para cada consola.

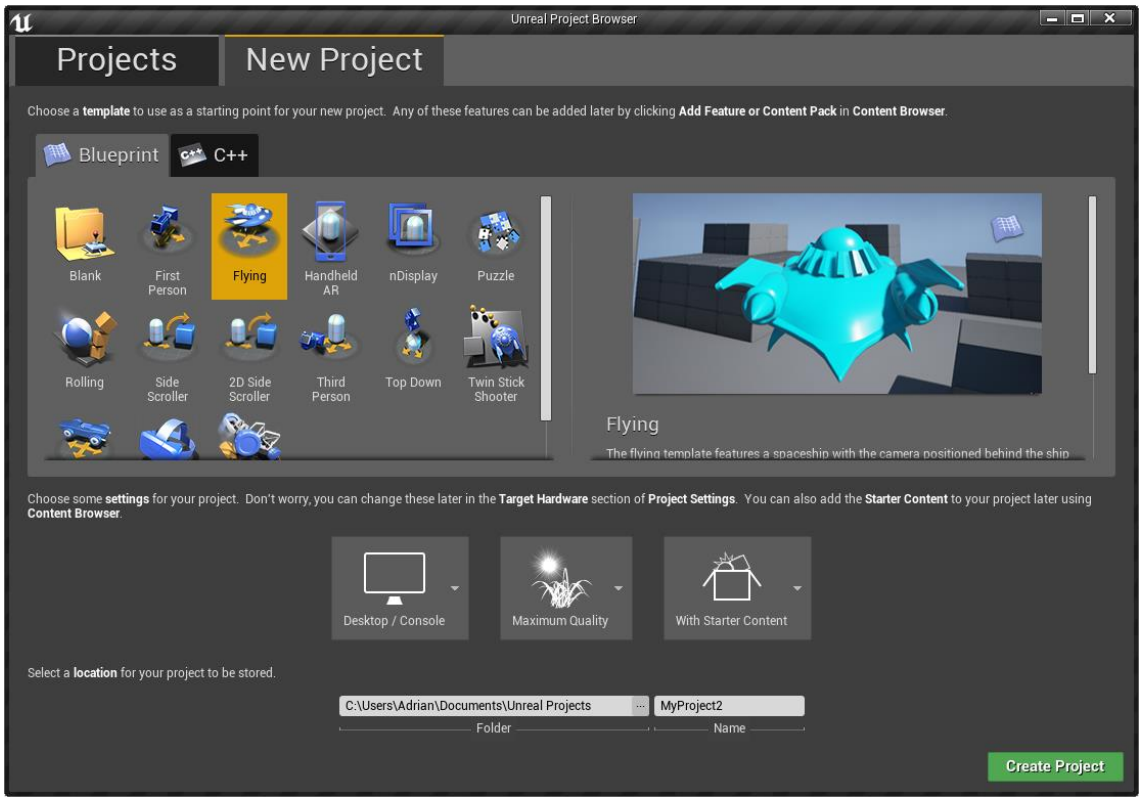

Fig. 26 Menú de crear Proyecto en Unreal.

Al haber seleccionado la plantilla de vuelo (*flying template*), veremos como el motor, tras unos minutos, abrirá el *Unreal Editor* nuestra herramienta de edición y desarrollo, con un juego pre-creado que consta de una nave espacial y unas cuantas estructuras negras. Si lo ejecutamos podemos comprobar que, además del mapa, Unreal ha generado por defecto un

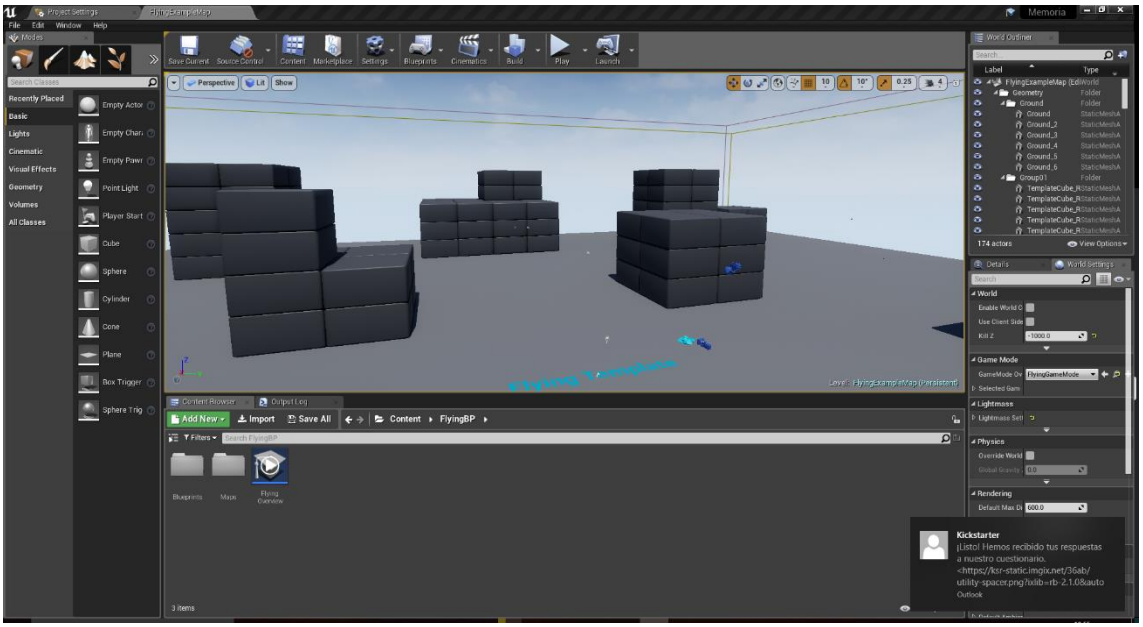

Fig. 27 Proyecto Vacío creado con la plantilla de Vuelo

esquema de control y unas mecánicas de vuelo predeterminadas, de las cuales nos desharemos en su mayoría por no ser relevantes para nuestro proyecto.

b. Creación de los niveles.

Lo primero que vamos a hacer es separar el juego en los niveles necesarios. En el Motor Unreal cada "Nivel" es un mapa. El nivel, puede tener o no un entorno generado desde el panel *Modes* desde ese panel, podemos generar un suelo para el entorno, que después, con distintas herramientas, podemos modificar a nuestro gusto, sin embargo, esa funcionalidad no es necesaria para nuestro prototipo.

Para mejorar la carga inicial del juego, y separar los menús de nuestro Nivel de juego, vamos a crear dos nuevos niveles a parte del nivel por defecto. Para hacer esto, buscaremos en el *Content Browser* la carpeta FlyingBP y dentro de ella la carpeta Maps, ahí está creado el nivel por defecto del juego, que renombraremos a Nivel. Además, crearemos dos niveles más: MenuPrincipal y Victoria. Para crearlos, solo tenemos que hacer clic derecho sobre el *Content Browser* y hacer clic sobre Level.

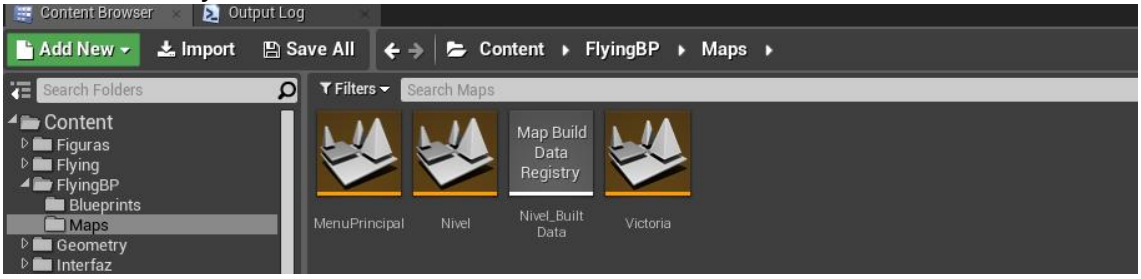

Fig. 28 Niveles del Juego

En Unreal, cada nivel puede tener una lógica asociada a él, y eso es lo que buscamos, especialmente para el Nivel de juego. Para mostrar los blueprints asociados al nivel, sólo se ha de pulsar en el botón de blueprints de encima del *Viewport* y, en la lista desplegable, pulsar sobre *"open level blueprint"*

c. Interfaces de usuario.

Ahora que tenemos los niveles creados y separados, vamos a empezar primero con las interfaces de usuario. Primero creamos una carpeta en la raíz del *Content Browser* a la que hemos nombrado Interfaz. Dentro de Interfaz, con el botón derecho del ratón, en la sección de *User Interface*, hemos creado un nuevo *Widget Blueprint.*

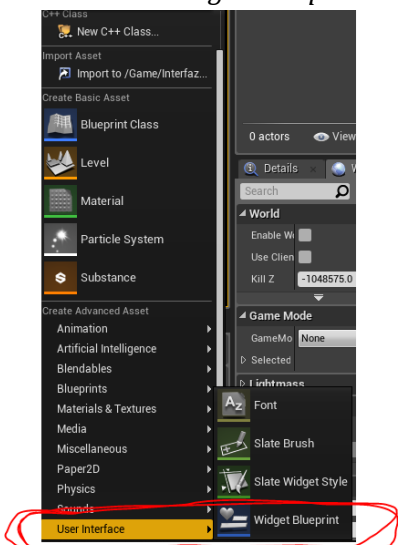

Fig. 29 Creación de un Widget

Los Widget son conjuntos de botones, paneles, imágenes, textos, y otros elementos de interfaz que podemos añadir a la pantalla en cualquier momento. Por ello son idóneos para crear Menús. Crearemos widgets para el menú principal, el menú de pausa, el menú de como jugar, el HUD del juego, los créditos del juego, y el menú de Victoria. También crearemos una carpeta para guardar las imágenes que usemos en dichos Widgets.

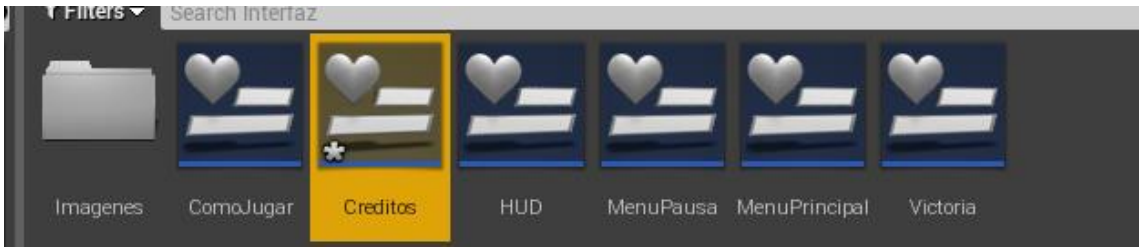

Fig. 30 Carpeta creada para la Interfaz de Usuario,

Menú Principal.

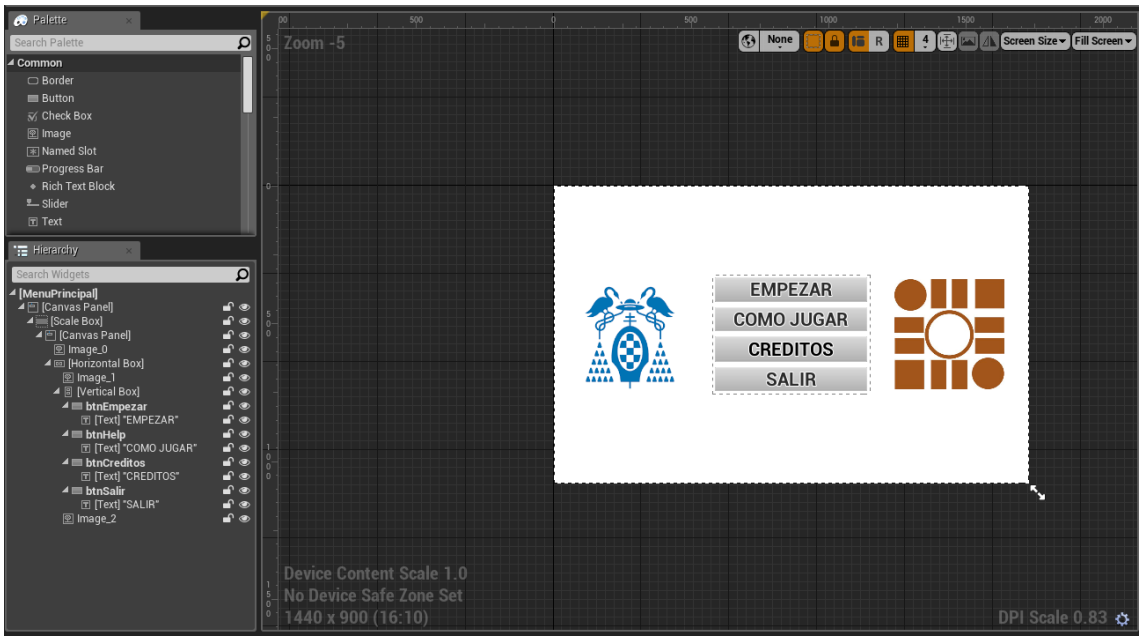

Fig. 31 Menú Principal

A la hora de crear el menú principal se ha optado por un diseño funcional, dado que no tenemos los conocimientos de diseño gráfico y de UX/I cómo para crear un menú más artístico. El diseño del menú es sencillo, pero se han tenido que hacer unas cuantas modificaciones para mejorar su funcionalidad.

El Menú cuenta con los botones ilustrados según los casos de uso CU-01 al CU-04, dispone de los botones de Empezar, Como Jugar, Créditos, y Salir.

Para diseñar el Menú, se han usado los paneles laterales de *Palette* y *Hierarchy*, estos paneles muestran las herramientas de las que disponemos para crear interfaces a nuestro gusto. Arrastrando cualquier elemento desde el panel de la paleta hacia el plano (donde se encuentra el diseño) o hacia la jerarquía y soltando dicho elemento podemos añadir elementos a la pantalla.

La interfaz siempre contará con un primer elemento *Canvas*, que puede tener todos los hijos que quiera. Si queremos que los menús del juego escalen con la pantalla, debemos meterlo todo a una *Scale Box* que se encargará de escalar el tamaño de sus contenidos a cualquier pantalla. Esto lo hace mediante unas propiedades de DPI, que son los puntos por pulgada que tendrá una imagen impresa. La escala de DPI aumentará el número de puntos a la razón por defecto conforme la pantalla sea más grande. Esta razón puede ser ajustada,

si bien, como ya hemos dicho antes, no tenemos los conocimientos de usabilidad y diseño de interfaces como para modificar dicha razón.

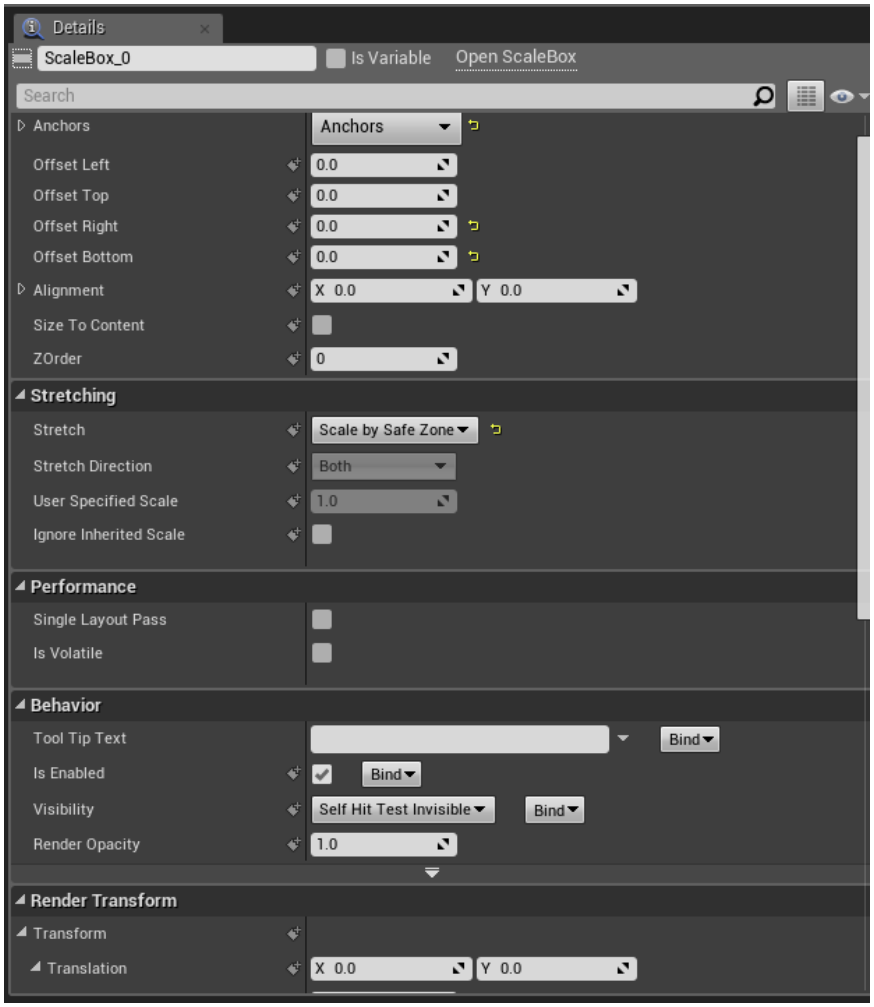

Fig. 32 Detalles de la ScaleBox

A la derecha de la pantalla sale el panel de detalles, en el cual podemos cambiar los atributos de cualquier elemento de nuestra interfaz. Es importante usar bien el atributo *Anchors* que es el que se encarga de fijar el sistema de Anclas, permitiendo crear un elemento fijado a cualquier punto relativo de cualquier pantalla (centro, arriba, abajo, llenándola, etc.).

La Scale Box, solo permite tener un hijo, de tal forma que no podríamos añadir todos los botones que quisiéramos, por eso, creamos un hijo de tipo *Canvas* sobre el cual podemos añadir todos los elementos que queramos.

Dentro de ese panel creamos una *Horizontal Box* que se encargará de ordenar nuestros elementos de forma consecutiva y horizontal. Dentro de esta caja incorporamos las imágenes de los logotipos de la Universidad de Alcalá de Henares y de la Escuela Politécnica Superior, ya que el prototipo se está realizando como proyecto de fin de grado en dicha facultad. También añadimos una *Vertical Box* que hace lo mismo que nuestra caja horizontal, pero en una orientación vertical.

Dentro de esa última caja metemos los botones y los textos dentro de estos. Los botones tienen una separación de 20 pixeles entre ellos, y se han modificado sus atributos para que escalen con la pantalla. Los textos dentro de los botones utilizan la fuente Roboto, que es la fuente por defecto de Unreal Engine, esta fuente nos sirve por ser una Sans Serifa. El tamaño de la fuente es de 50, en color negro, también se ha aplicado una línea blanca exterior a la fuente de 3 pixeles para mejorar su legibilidad.

Cómo Jugar

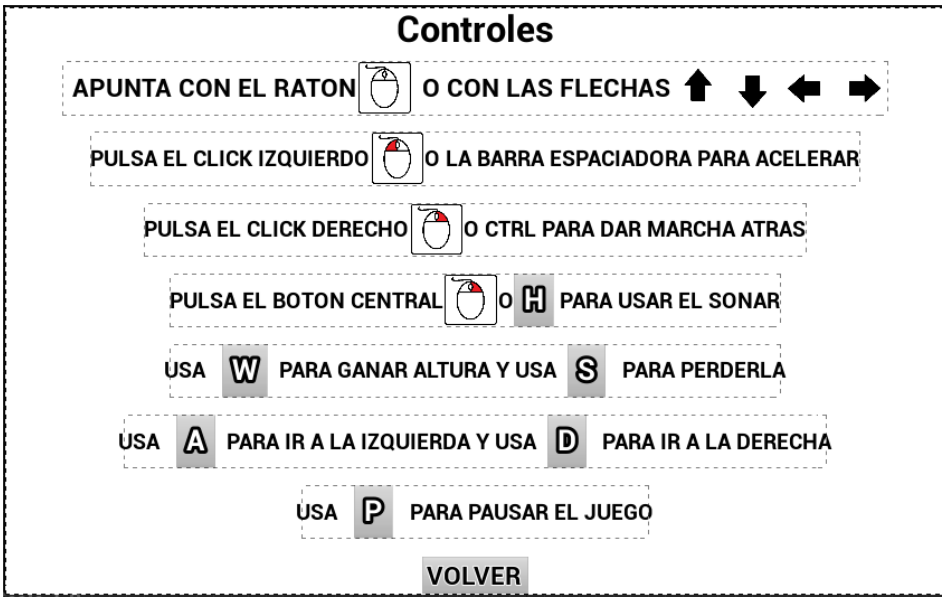

Fig. 33 Cómo Jugar

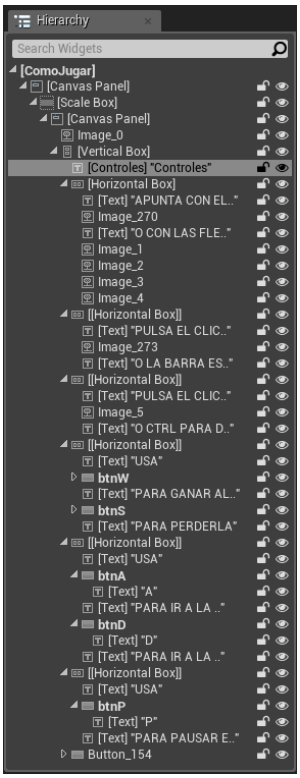

Fig. 34 Jerarquía del menú Cómo Jugar

Al igual que con el menú principal, el menú de Cómo Jugar es sencillo y funcional, si bien su diseño intenta mostrar la información de manera ordenada y teniendo en cuenta la accesibilidad.

El menú está diseñado de la misma forma que el menú principal, aprovechando el sistema de Anclas en conjunción con las *Scale Box* para crear una interfaz de usuario que puede ser visible en diversos tamaños de pantalla. El contenido esta ordenado en su totalidad dentro de una *Vertical Box* para que se mantenga ordenado de manera vertical.

Después, dentro de ese panel, se encuentran ordenados un conjunto de *Horizontal Box* que dentro contienen los textos, imágenes y botones necesarios para formar la pantalla. Los botones, de acuerdo con los requisitos funcionales, servirán para decir en voz alta la tecla asignada a cada entrada distinta en nuestro sistema de control.

Cabe destacar que las imágenes, al no encontrar imágenes claras que sirvieran para indicar mejor los controles necesarios, han sido realizadas de forma específica para este proyecto.

Por último, se encuentra el botón que nos permite volver al Menú Principal.

El texto “Controles”, que actúa a modo de título, tiene un color de fuente negro para que resalte sobre el resto de los textos, que se han puesto con un reborde negro de 4 pixeles, usando la fuente Roboto en un color negro que contraste con el fondo blanco del menú. Las letras que se encuentran dentro de los botones tienen un espaciado interno de 10 pixeles que permite que no se vean apelotonadas con el resto de texto.

La pantalla contiene mucha información, sin embargo, esta se encuentra espaciada, y expresada de la forma más clara que se ha podido. Si hubiera que añadir más controles, sería recomendable que fuese un menú paginado, en el que el jugador pudiera avanzar a su propio ritmo por la página.

Victoria y Créditos

Fig. 35 Pantalla de Victoria

Las pantallas de Victoria y de Créditos son las más sencillas, mostrando solo la información indicada con un botón. La información, al igual que en el menú de Cómo jugar, esta resaltada con negro para que sea más legible.

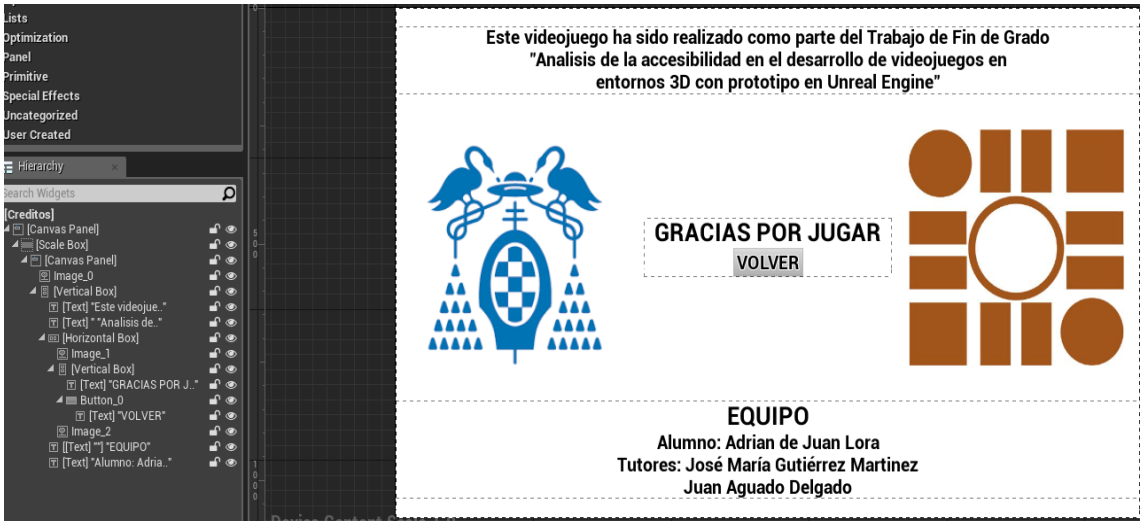

Fig. 36 Pantalla de Créditos

HUD

Fig.37 HUD

El HUD (*Head-Up Display*) de un videojuego, comúnmente denominado barra de estado, es un elemento de interfaz de usuario que se muestra mientras se ejecuta el juego. El HUD sirve para comunicar información al jugador sin necesidad de pausar el juego, dinamizando así la experiencia. En nuestro caso, el HUD es muy sencillo. Debe indicar en cada momento el objetivo del juego, así como en número de rocas que se han recogido. Como se han planificado 5 rocas, ese texto puede ser fijo, sin embargo, el texto que informa de la cantidad de rocas recogidas ha de ser modificado en tiempo dinámico, esto es posible gracias al atributo *Bind* o vínculo de un elemento.

Con la vinculación podemos atribuir el resultado de una función a un texto u otros atributos determinados de ciertos elementos, mostrando así información cambiante en el tiempo de ejecución. En este caso, el texto CantidadRocas, irá vinculado a una función que devuelva el número de rocas recogidas.

En esta interfaz no se ha utilizado el elemento ScaleBox, ya que este dificulta la colocación estática de ciertos elementos por la pantalla, sin embargo, sí que se han anclado los componentes para que salgan centrados. El texto es blanco, con un reborde negro para asegurar así que siempre es legible durante el juego.

d. Materiales, Modelos y Sonidos.

Unreal Engine permite la creación de materiales, de modelos y de sonidos. En este apartado veremos que clases de cada tipo han sido creadas con la intención de mejorar la accesibilidad.

Materiales

Los materiales, definen que apariencia tienen un modelo 3D. Aunque un modelo 3D pueda tener color, no tiene textura ni material, da la apariencia de ser un objeto liso. Sin embargo, con los materiales, esto se puede cambiar. Los materiales permiten cambiar cualidades

estéticas y asignárselas a un modelo 3D, un material puede ser una imagen (que se repetirá en bucle hasta cubrir el objeto) o un solo color. Un material define también cualidades como la rugosidad que presentara un objeto, o lo "metálico" que aparenta ser, es decir, cuanta luz refleja.

Dado que Unreal Engine no nos permite tener un mundo sin luz (pues nada se vería si no se reparten luces estáticas), es importante, para cumplir con el RF07, que asignemos los colores y los materiales con cuidado.

Para asegurarnos de que el juego se vea en alto contraste, lo mejor es diseñarlo en una escala de colores azul y naranja, y asegurarnos de que estos varían en brillo. También, marcaremos con rebordes negros los objetos sólidos, para que sea más fácil distinguirlos. Nuestra nave será verde claro, dado que así contrasta con el azul oscuro que tendrán los edificios. El suelo, será completamente naranja. Las rocas, que serán los objetos que recoger, serán blancas y muy brillantes, para que llamen la atención.

Así pues, hemos de diseñar distintos materiales que podamos asignar a cada uno de estos objetos.

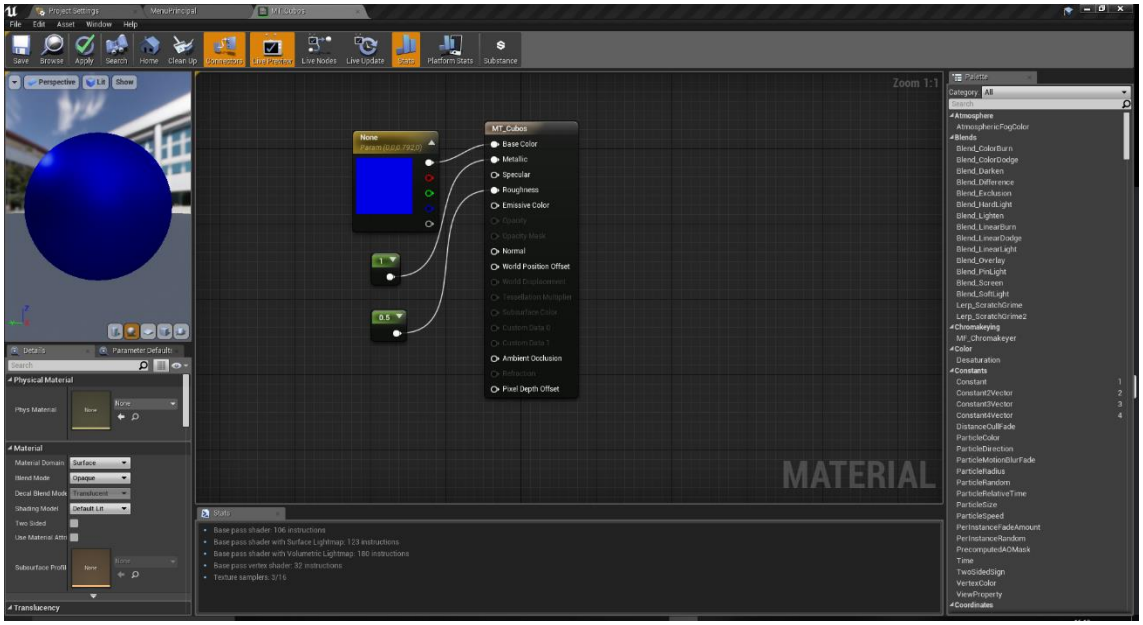

Fig. 38 Creación del material para los edificios

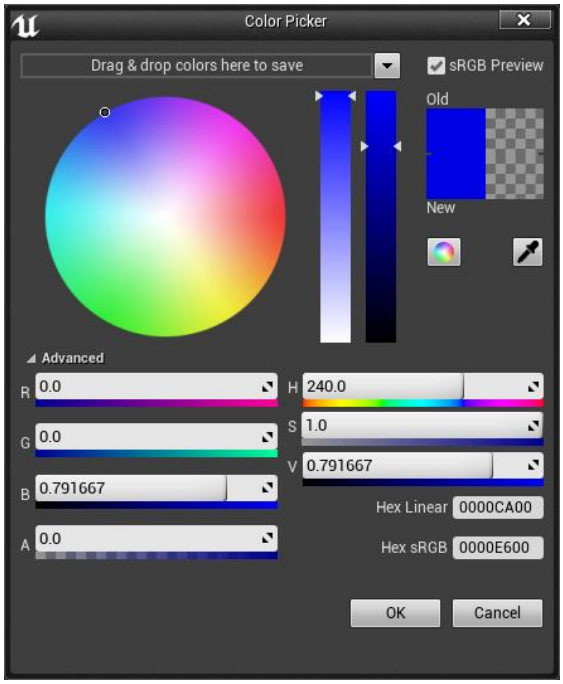

Fig. 39 Selector de color

Cuando vamos a crear un material, se nos muestra la pantalla que se aprecia en la figura 38. Podemos ver que abajo a la izquierda, se encuentra un panel de detalles que nos permite alterar ciertas propiedades del objeto. A la izquierda se nos muestra un Viewport con una esfera que tiene nuestro material aplicado, esto sirve para poder ver de forma rápida como afectan los cambios que hagamos a nuestro material.

A un material se le debe asignar un color o una imagen, en nuestro caso, para asignarle un color, solo tenemos que hacer clic derecho sobre el sistema de Blueprints y seleccionar la opción *Vector Parameter*, en los planos de materiales, los vectores representan las cualidades RGB que tendrá el material. Cuando hacemos doble clic sobre el vector (El nodo con el cuadrado azul en la figura 38), podemos ver como se abre un sistema de selección de color, que nos permite elegir qué color deseamos formar.

Este vector creado, debe ser asignado al atributo BaseColor del material. También es importante crear constantes numéricas entre 0 y 1 para asignárselas a los atributos Metalic y Roughness, estos atributos se encargan de representar cómo de metálico se ve un objeto y cómo de suave parece. Calibrando bien estos dos últimos atributos, podemos manejar cuánta luz refleja un objeto.

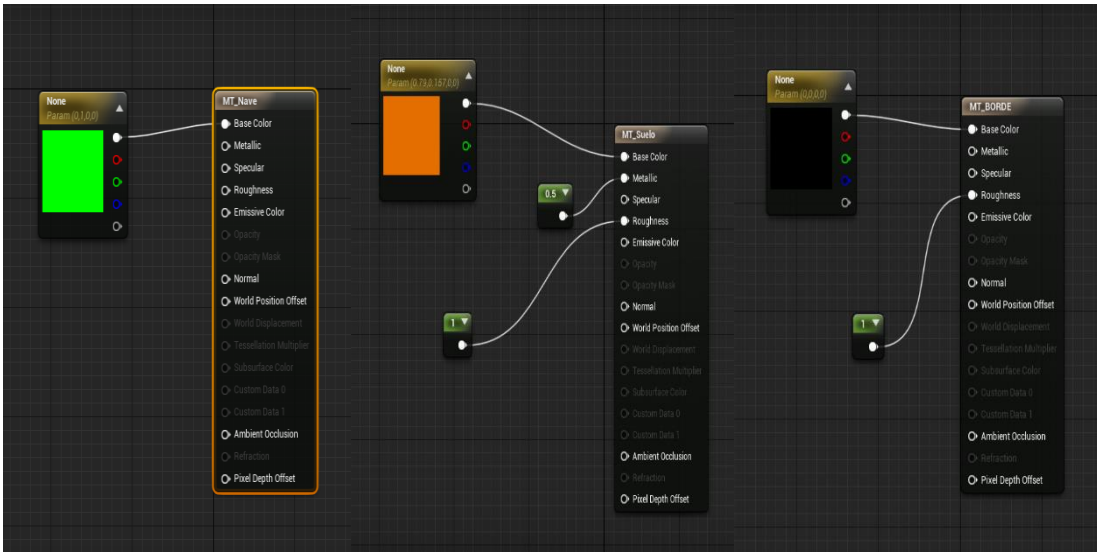

Fig. 40 Materiales de la Nave, el Suelo y los bordes

Si bien es recomendable dejar la creación de materiales a diseñadores, ya que entienden mejor las cualidades del color, en el equipo se ha realizado un esfuerzo por hacer colores de alto contraste que cumplan bien su propósito. El azul para los edificios es oscuro y metálico y rugoso, para que dé la impresión de ser un objeto duro y que pueda diferenciarse del cielo. El naranja del suelo tiene una situación similar al azul, sin embargo, parece más áspero para dar una sensación de tierra/asfalto. El verde de la nave es brillante para que contraste con el resto de los colores. Y el Negro de los bordes es oscuro y no refleja ninguna luz para asegurarse de que siempre resalta el objeto que rodea. Las rocas no tienen material, ya que un objeto sin material se queda con un material blanco, metálico y brillante, que es lo que buscamos.

Modelos

Los modelos 3D han de ser creados desde un editor de modelos 3D, sin embargo, Unreal ofrece ciertos modelos de figuras geométricas básicas y modelos comunes que vamos a aprovechar para nuestro prototipo. Los modelos creados por Unreal o un editor 3D son llamados StaticMesh, estos modelos son estáticos, es decir no cambian de forma. Estos modelos, pueden ser modificados de una forma limitada, aplicándoles texturas y materiales por defecto y cambiando su tamaño. Es importante notar que, si se cambia un atributo de un modelo instanciado, ese atributo solo se cambia en ese modelo, sin embargo, si se cambia en la clase que contiene el plano del modelo 3D, que está en el *Content Browser* esos cambios se producen en todas las instancias futuras del modelo, y en todas las instancias pasadas que no hayan sido modificadas.

Aparte de los StaticMesh, que han de ser importados, el desarrollador puede crear actores, que como ya hemos dicho son objetos destinados a interactuar con el usuario. Los Actores, pueden contener StaticMesh, sonidos, partículas, colisiones e incluso cámaras.

En nuestro caso, hemos modificado los Static Mesh del suelo y de los cubos, para cambiarles las texturas, y poder crear el mapa que nosotros queramos.

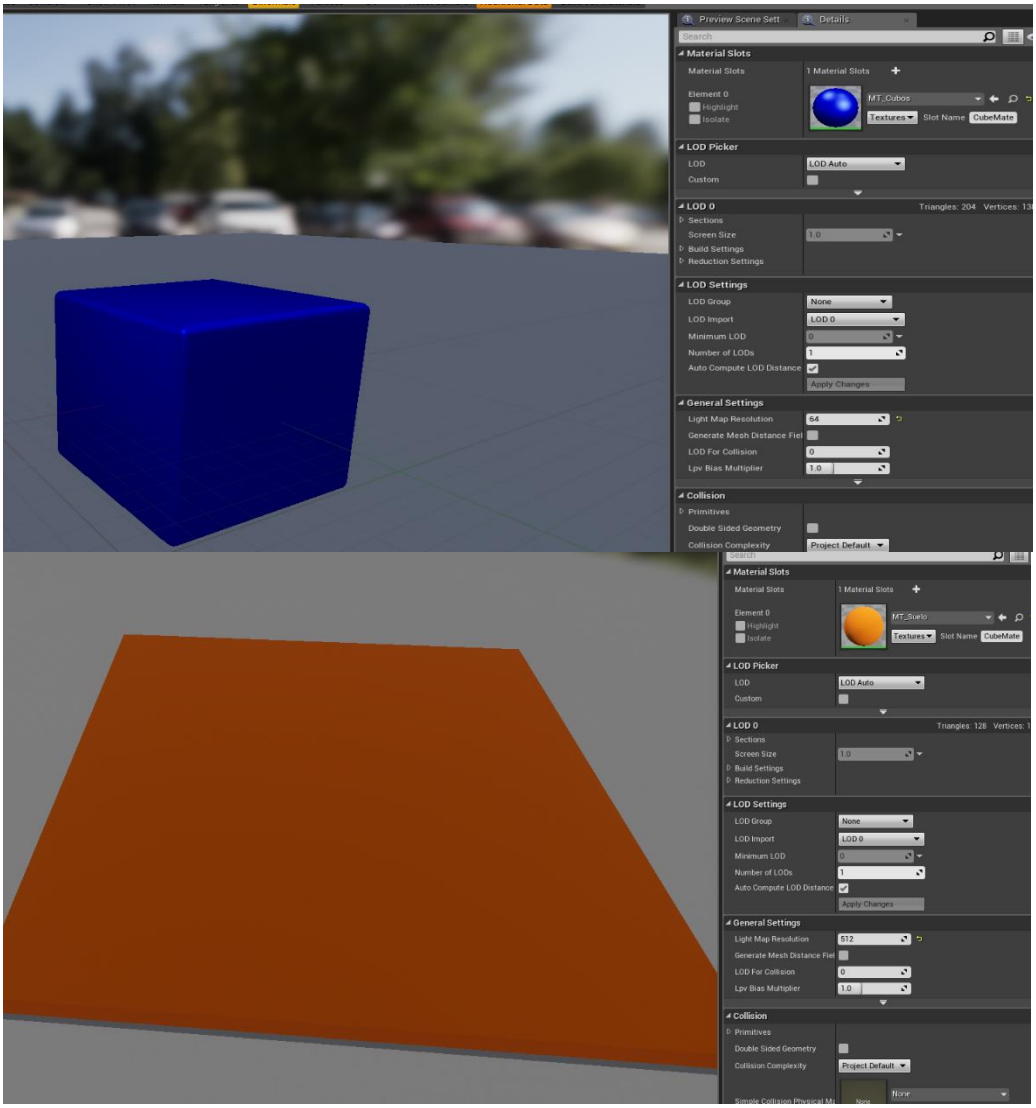

Fig. 41 Static Mesh de los cubos y del suelo modificados

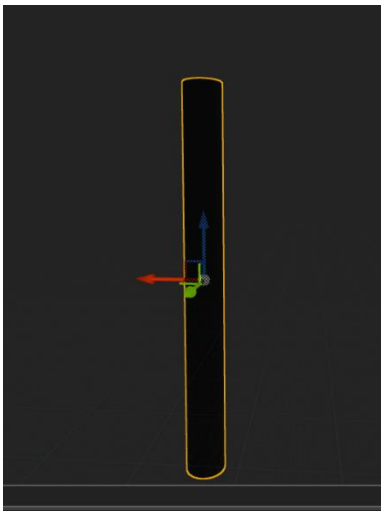

Fig. 42 Borde

Para crear los bordes, hemos creado un nuevo actor desde el menú contextual del *Content Browser* pulsando en la opción de nuevo plano. A este actor solo le hemos asignado un static mesh de cilindro, que hemos estrechado, y al que le hemos asignado el material del borde. Al no asignarle ninguna funcionalidad a este actor, no interactuará con el usuario, pero seguirá cumpliendo su función estética.

Lo más interesante ha sido la creación del modelo de las Rocas. Para crearlas, al igual que con los bordes, hemos creado un actor desde el modelo de roca que venía con los archivos por defecto de Unreal. Sin embargo, a esta roca le hemos

quitado la textura. Después, le hemos añadido una colisión en capsula, y nos hemos asegurado de que su tamaño sea similar al de la roca. Esta colisión es de suma importancia porque es la que se encargará de detectar al jugador y permitir que este recoja las rocas. Por último, le hemos añadido un sonido, asegurándonos de que tenga atenuación espacial en esfera (tocando las propiedades del sonido), y poniéndole un radio de atenuación adecuado. Con la atenuación en esfera y la propagación espacial, Unreal se encarga de hacer Ray Casting, y asegurarse de que el sonido este localizado alrededor del objeto, y pueda ser ubicado con el balance de sonido que dispone el sistema (esto sólo funciona con cascos).

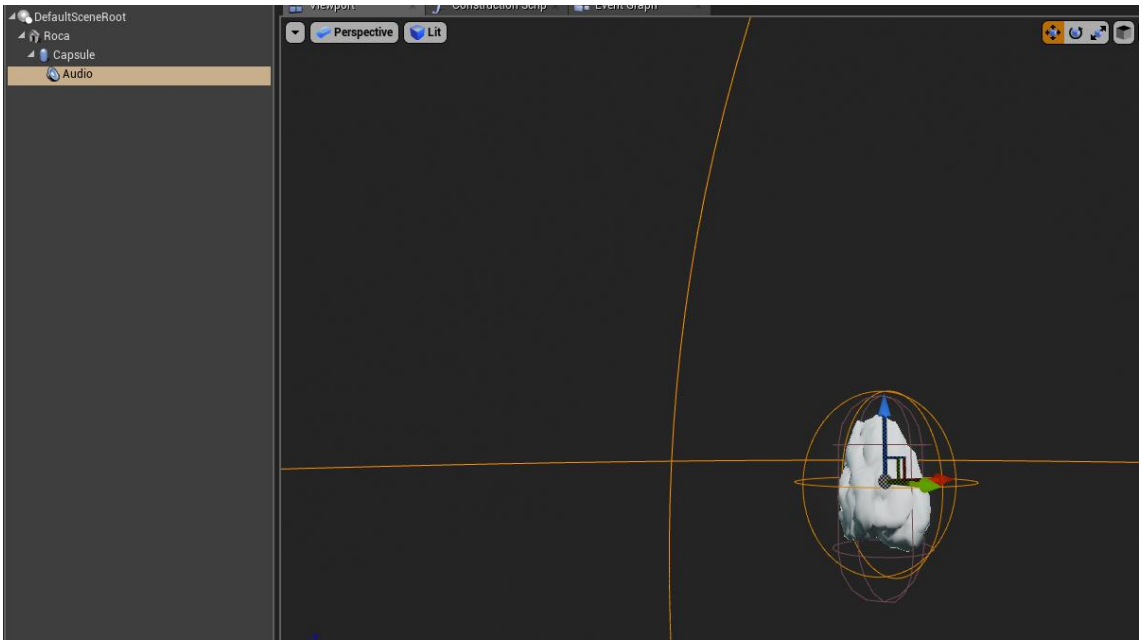

Fig. 43 Diseño de la roca, Árbol de componentes.

Sonido

Unreal nos proporciona ciertos sonidos por defecto, algunos, como el fuego cuando la roca se encuentre cerca o el sonido de derrumbe al chocar con un edificio, han sido utilizados por nosotros. Sin embargo, es imposible encontrar los sonidos necesarios siempre, ya que Unreal solo permite importar sonidos en formato .WAV de 16 bits. Algunos de los sonidos han sido descargados de la librería online: https://www.pacdv.com/sounds/voices-1.html

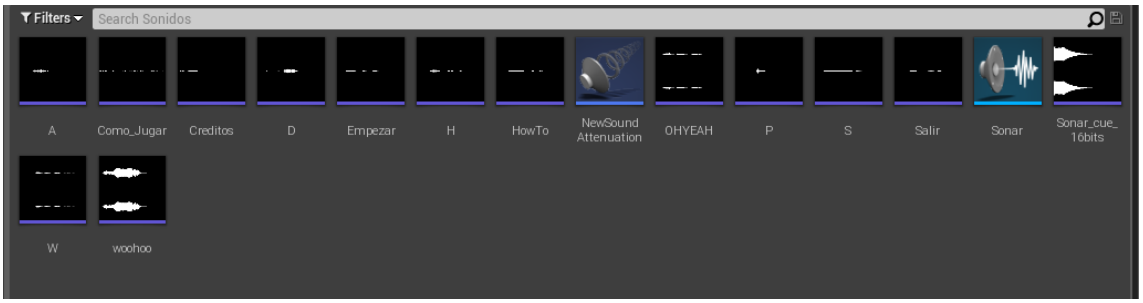

Fig. 44 Sonidos del juego.

Para facilitar el uso de sonidos, la mayoría de estos han sido grabados con el programa Audacity, y después se han añadido al proyecto. Esto ha sido especialmente útil para grabar

un TTS (Text To Speech, o lectura de un texto) para el menú de como jugar, dado que Unreal no permite el uso de lectores de pantalla.

El sonido que más trabajo ha llevado dentro de Unreal, ha sido el sonido del sonar. Unreal permite trabajar con sonidos, cambiando sus propiedades y usando herramientas de un mezclador. Esto permite a cualquier ingeniero de sonido modificar sonidos para usarlos de manera eficaz y artística dentro del juego. En nuestro caso, dado que no disponemos de dicha experiencia, hemos tenido que crear el sonido del sonar con un archivo de audio descargado de la página: https://freesound.org/people/SpankMyFilth/sounds/215206/ de manera gratuita. Por las propiedades que debe tener el Sonar, este sonido debía de ser incorporado con una atenuación espacial del radio del mapa, que permitiera escucharlo desde cualquier parte de este, para poder localizarlo en cualquier momento. Para ello, se ha creado un nuevo archivo de sonido con Unreal (desde el menú contextual del *Content Browser* se ha creado una nueva Sound Cue). A dicho archivo, se le ha incorporado este sonido y se le ha aplicado la atenuación espacial necesaria

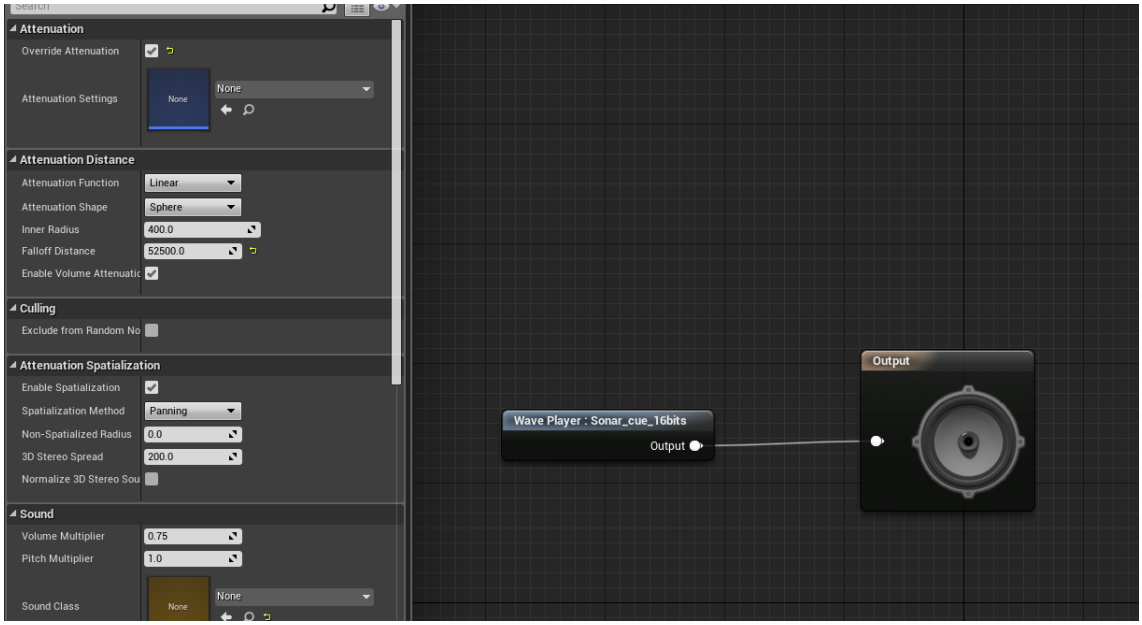

Fig. 45 Sonido del Sonar

e. Diseñando el nivel.

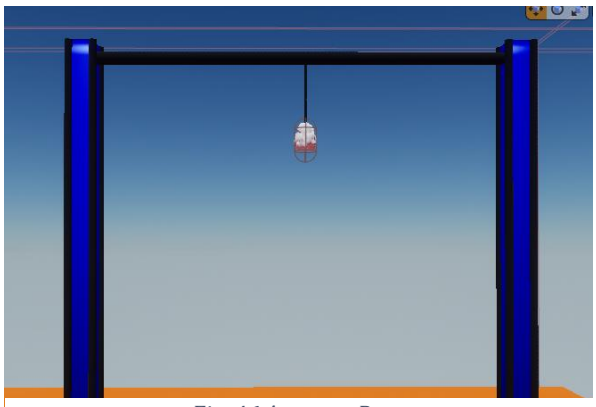

Fig. 46 Arco con Roca

El nivel, que por defecto dispone de ciertos objetos colocados en un mapa por el que volar, queremos que sea modificado por nosotros. Hemos decidido separar el nivel en 5 zonas. La ciudad inicial, que se hará modificando el mapa por defecto que ofrece Unreal. Cerca de la ciudad se pondrá una pirámide, que tendrá la primera roca en su cima. También habrá unas torres con una roca entre medias, en un hueco relativamente

estrecho. Un estadio, con una roca dentro, un túnel con una roca en su centro, y un arco con una roca colgando.

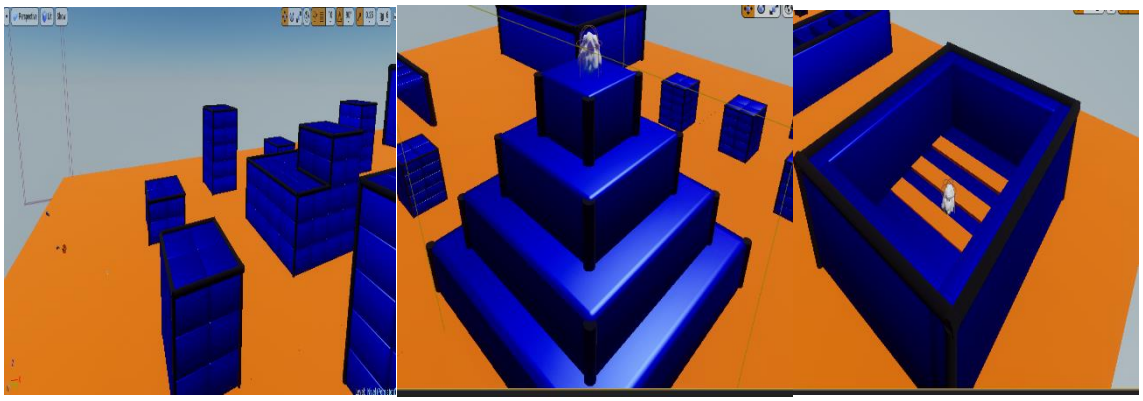

Fig. 47 Ciudad, Pirámide, Arco

Todas las estructuras estarán construidas con el cubo azul, y resaltadas con rebordes negros. Si fuera necesario, se usaría el suelo naranja para marcar contraste dentro de una figura.

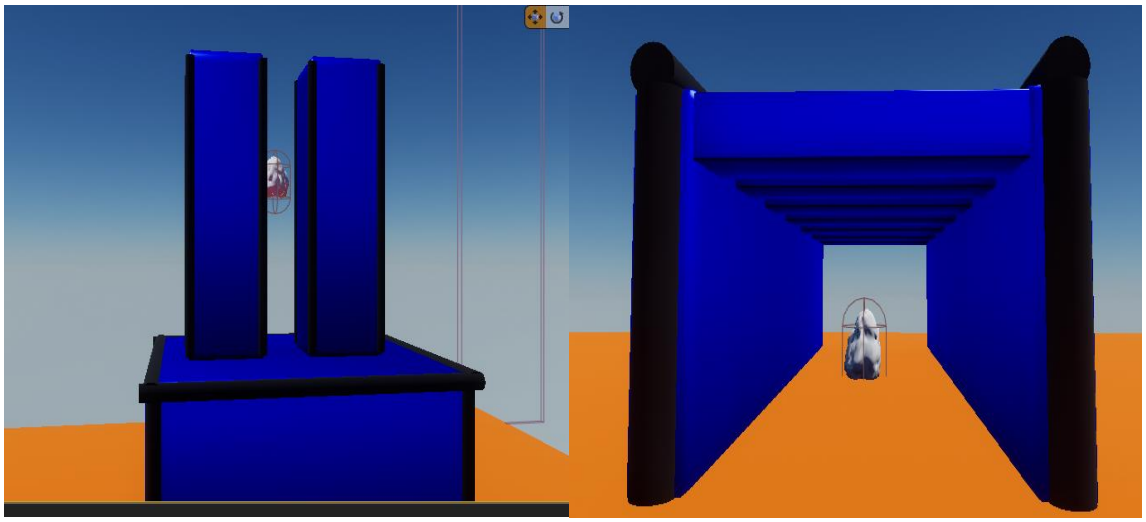

Fig. 48 Torres y Túnel

También se han añadido volúmenes bloqueantes en los límites del nivel, se pueden apreciar al fondo de la figura 47. Estos volúmenes son colisiones invisibles que bloquean el paso al jugador, previniendo que salga de la zona de juego.

f. Añadiendo funcionalidad.

En este apartado se detallarán las funciones creadas con el sistema de planos de Unreal para el funcionamiento del juego. El sistema de planos de Unreal permite tener funcionalidad sobre gran cantidad de clases distintas, no solo sobre el nivel del juego. En nuestro caso, hemos añadido funcionalidades al propio nivel de juego, a las interfaces de usuario, al peón que controla el jugador y a las rocas que son el objetivo que recoger.

Programando el sistema de control.

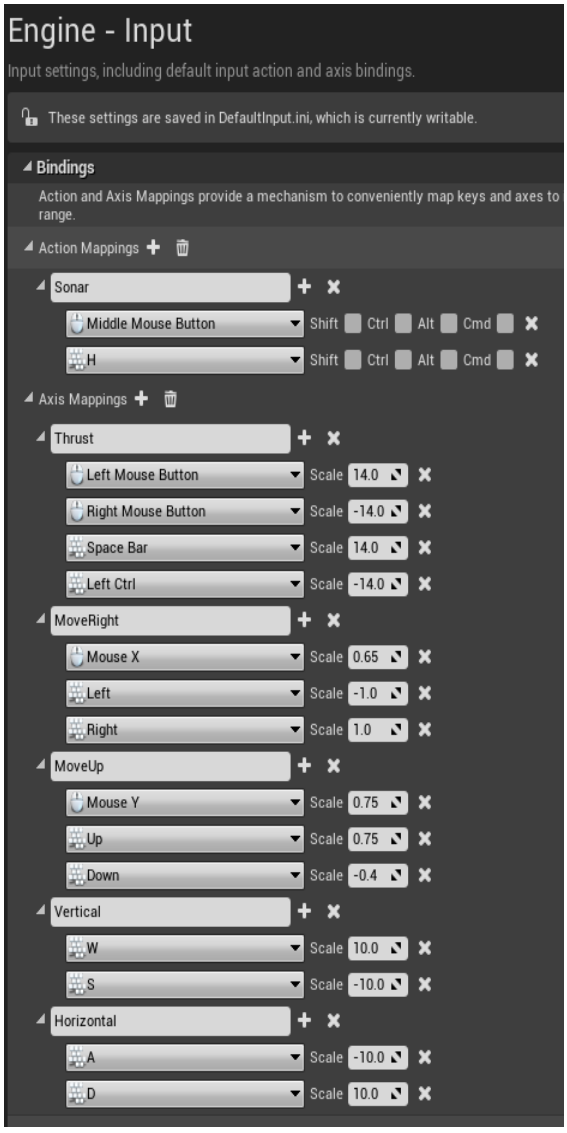

Fig. 49 Esquema de control del Prototipo

Lo primero y lo más necesario es dejar claro que entradas va a reconocer nuestro programa. Estas entradas pueden ser definidas desde la configuración del proyecto en *Project Settings -> Input*. Esta configuración solía ser estática, no pudiendo cambiar los controles predefinidos con la versión base de Unreal Engine, era necesaria la instalación de complementos o modificación de librerías del motor para poder realizar cambios en el esquema de control. Sin embargo, desde la versión 4.17, es posible cambiar este esquema, permitiendo la entrada del usuario. Por desgracia, al ser una funcionalidad relativamente nueva no funciona bien a una escala completa del esquema de control, y por tanto no será utilizada.

Existen dos tipos de entradas que detecta el juego. Axiales, y de acción.

Las entradas Axiales son aquellas que se mueven en una escala, son normalmente utilizadas para el movimiento de los peones u otro tipo de acciones que han de ser progresivas. Funcionan particularmente bien con dispositivos que permitan dicha escala, como los ejes de movimiento de un ratón o los Joystick de un mando. Como se puede apreciar en la Figura 49, existen tres componentes en una entrada axial: El nombre, el control y la escala. El nombre es el nombre de la acción a realizar, cuando se quiera invocar dicha acción desde el sistema de planos, se invocará el evento refiriéndonos al nombre de esta, por ello es importante elegir un nombre que defina bien la acción a realizar.

El control es la entrada asociada a dicha acción, como se puede ver existen variedad de entradas distintas para el control, desde las teclas de cualquier teclado, a botones y ejes del ratón. Si se cuenta con los SDK de desarrollo para distintas plataformas se pueden añadir esos controles también, por ejemplo, con el SDK de Nintendo Switch, se podría asignar una acción con el giroscopio de los mandos.

La última parte, la escala, implica entre que rango de valores puede tener ese control, siempre desde 0 al máximo de la escala. Esto puede ser modificado para cambiar la sensibilidad de los controles. Un control que solo pueda tener dos estados, como una tecla, siempre estará en el 0 hasta que se pulse el control, en ese momento cambiará a su valor

máximo. Los controles que, si dispongan de un eje, por ejemplo, un Joystick, se detectará la posición de dicho control respecto al centro, siendo solo necesario añadir una escala. Por ejemplo, si se le asigna el control del eje Y de un Joystick a la acción Vertical con una escala de 10, este estará en 0 cuando el control no esté siendo tocado, en 10 cuando se mueva hacia arriba, y en -10 cuando se mueva hacia abajo. Por último, el ratón funciona de forma diferente al resto de entradas axiales, dado que, en lugar de detectar su posición respecto a un centro, detecta la variación de la posición respecto a la última posición en la que haya estado quieto. Siendo siempre hacia la derecha y hacia arriba las variaciones positivas en su eje X e Y respectivamente.

Los controles de acción son más sencillos al no presentar variación en una escala. Solo se detectan dos estados: Pulsado y Soltado. El primero manda una señal de activo siempre que la entrada este siendo pulsada, el segundo manda una única señal cuando esta se suelta. Por desgracia, Unreal no tiene una opción para dos pulsaciones consecutivas, esta opción puede ser programada mediante temporizadores internos, pero los temporizadores se refrescan con los frames del juego, y puede dar problemas en entornos de bajo rendimiento.

El prototipo por defecto de Unreal traía los controles de “Thrust”, “Move right” y “Move Up”, solo pudiendo usarse con el ratón, estos han sido modificados y añadidas distintas entradas y más acciones con motivo de la accesibilidad. Sonar es la entrada dedicada a activar el sonar, Vertical es la entrada dedicada a ganar o perder altura y Horizontal es la dedicada a desplazarse hacia a la izquierda o a la derecha desde el objeto.

Una vez puestas las entradas, es necesario programar que efectos tienen en el juego. La práctica más recomendable es programarlas dentro de la clase del peón del jugador. En nuestro caso, en el FlyingPawn.

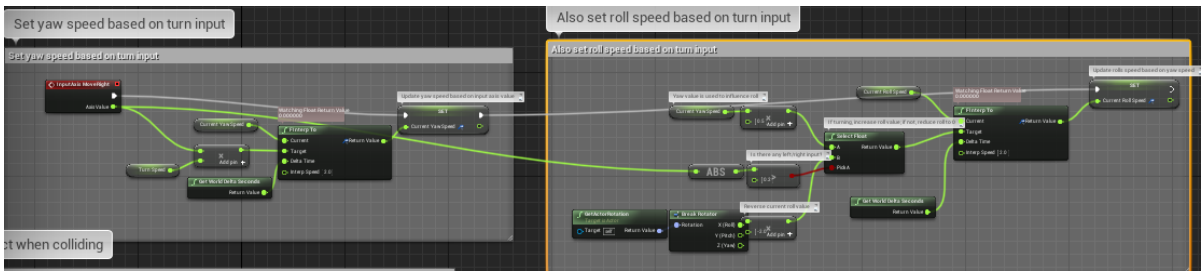

Fig. 50 Funciones de Giro y equilibrio del Peón

Unreal tenía programadas por defecto ciertas funciones que se han mantenido por su utilidad. Las mostradas en la figura 50, se encargan de mover la nave en la dirección en la que se mueva la entrada MoveRight, para ello incrementa dentro de unos límites de velocidad máxima, el giro de la nave, desplazando así su vector de movimiento frontal, también aplica una rotación a la nave para dar una impresión más fluida de movimiento, si la nave esta quieta, devuelve esa rotación a un número cercano a cero, estabilizando la nave y poniéndola en una posición paralela respecto al plano. Existe otra función similar para la inclinación vertical, solo que cuando se produce esa rotación la nave no se estabiliza para poder cambiar la altura fácilmente. Sin embargo, estas funciones no hacen el propio movimiento, solo cambian unas variables de giro y velocidad dentro de la nave. La actualización de la posición se hace con esas variables en la función que se muestra en la

figura 51. Con los nodos AddActorLocalOffset y AddActorLocalRotation se cambia la rotación del actor tomándose el propio actor como referencia, existen otros nodos que

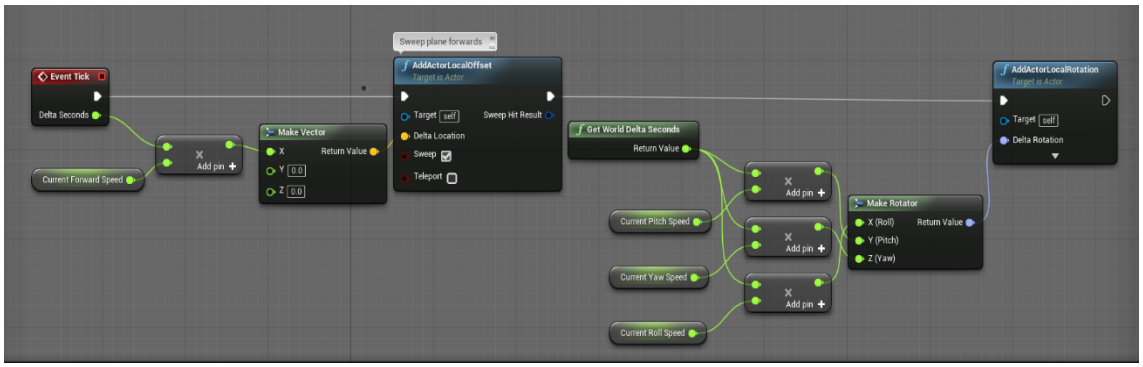

Fig. 51 Aplicación de la velocidad de Giro y Frontal

toman el centro de coordenadas del mapa como referencia. Ese cambio se hace creando un vector de movimiento y de rotación con los módulos MakeVector y MakeRotator, que se hacen con los valores previamente calculados.

Por nuestra parte, se han modificado los controles de aceleración, y añadidos controles para los movimientos Verticales y Horizontales.

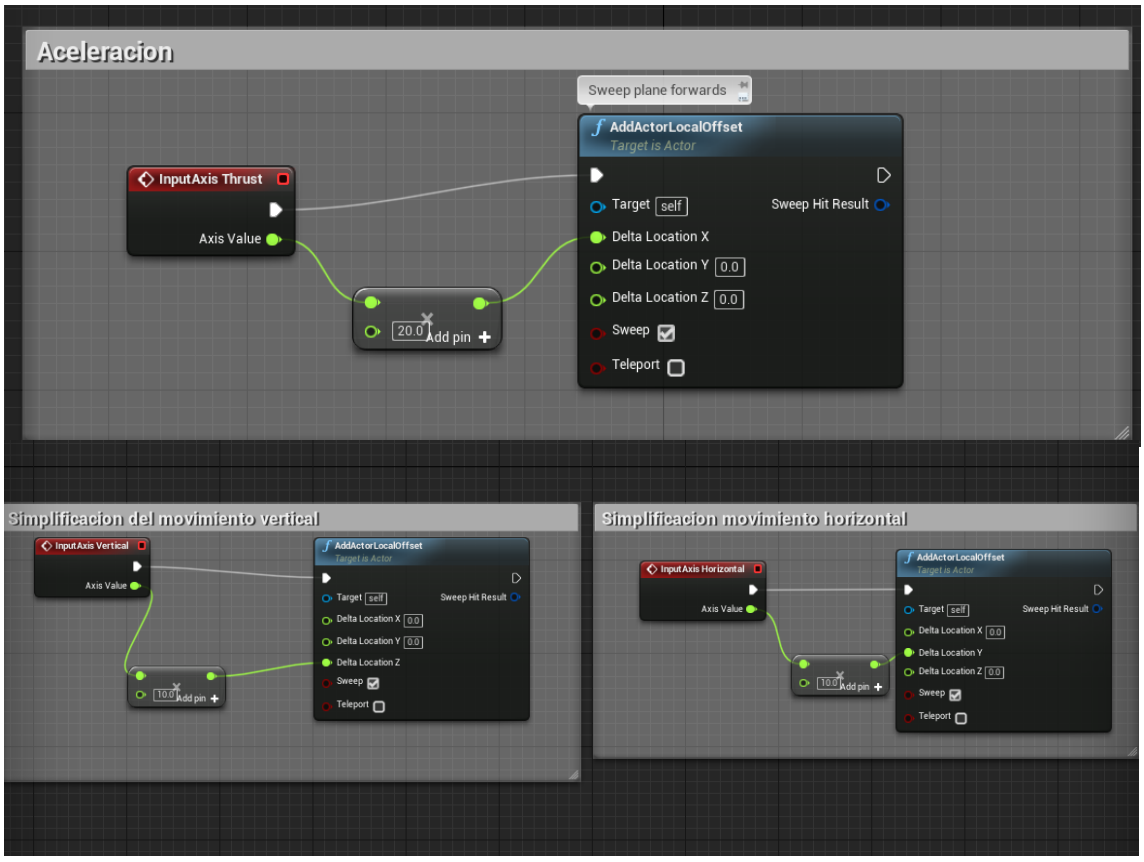

Fig. 52 Aceleración, movimiento vertical y horizontal

Dado que los movimientos de aceleración, verticales y horizontales ocurren cada uno en plano, son muy similares, simplemente cogemos el resultado de la entrada axial, y lo multiplicamos por un factor para que se mueva de manera más rápida por el mapa. AddActorLocalOffset, como ya se ha explicado antes, desplaza al actor de forma relativa, usando al propio actor como eje, sin embargo, es importante remarcar el uso del parámetro Sweep, que hemos activado en todos los movimientos, este parámetro hace que este movimiento ocurra deslizándose, en lugar de teletransportándose. Si no se activa este parámetro, no se producen golpes en la colisión, y el actor puede atravesar otros objetos.

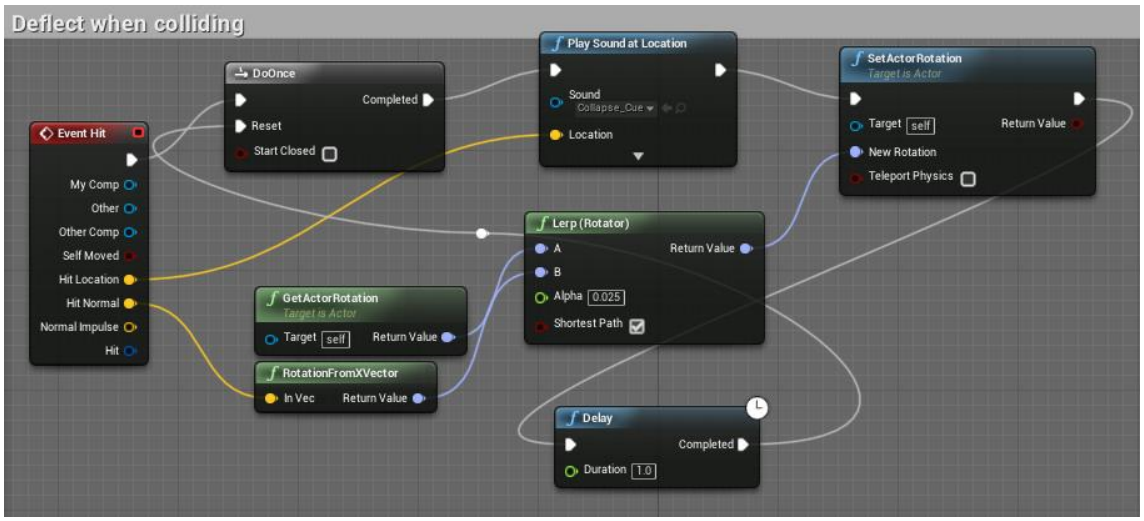

Fig. 53 Control sobre la colisión.

En cuanto al sistema de colisión, el proyecto pregenerado tenía una función para que el objeto se desviara cuando colisionaba con algo, nosotros hemos modificado esa función para que, además de desviarse, reproduzca una pista de sonido que indique que se ha producido dicha colisión. Esto se ha hecho con el nodo PlaySoundAtLocation, que se encarga de instanciar un sonido en una posición. Usamos el nodo DoOnce, para asegurarnos de que no se instancien varios sonidos al chocar deslizarnos por la superficie de un objeto, este nodo, se asegura de que la acción solo se realice una vez, y, junto al nodo Delay, que introduce una espera de un segundo, el nodo DoOnce ejerce como una especie de semáforo, haciendo que solo se pueda reproducir el sonido cada segundo.

El control del sonar está en los planos del nivel, ya que usa información relativa a este y se explicará más adelante en este mismo documento.

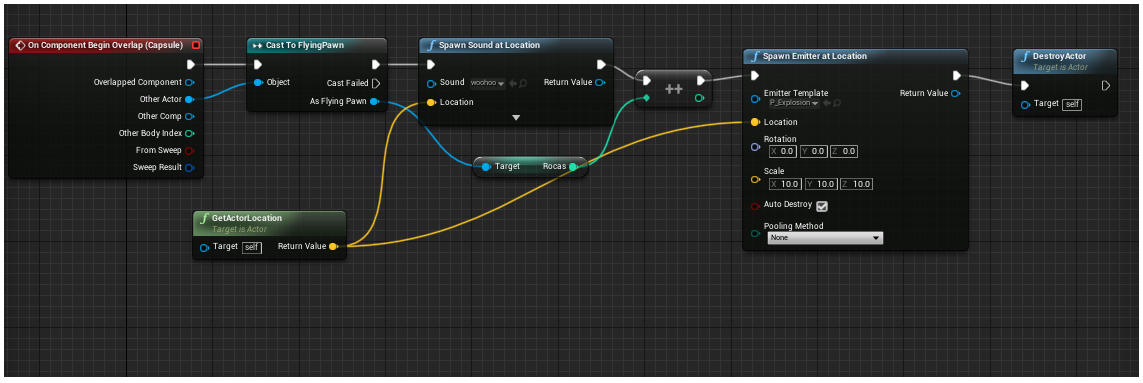

Fig. 54 Control de Recolección de Rocas

Por último, el control de recolección de las rocas, que figura dentro del actor de la Roca. En esta función, lo primero que hacemos es uso del nodo CastTo, que intenta hacer una conversión de un actor de tipo object, a un actor especifico. En este caso, la conversión se hace con el actor que colisiona con la roca, y se hace una conversión a FlyingPawn, nuestro peón. Si esta conversión es válida, se aumenta en uno la variable "Rocas" del actor, que es de tipo Integer, y lleva la cuenta del número de rocas que han sido recogidas en el nivel. También se hace uso de los nodos SpawnSoundAtLocation, que es similar a PlaySoundAtLocation, pero nos devuelve la instancia del sonido para poder modificarla, y del nodo SpawnEmitterAtLocation, que reproduce un efecto de partículas en una zona. En este caso, se reproduce un sonido Wohoo, que suena como una celebración, y unas partículas de explosión para indicar que se ha recogido la roca. Por último, se destruye el actor de la roca para que desaparezca del mapa.

Programando el Nivel

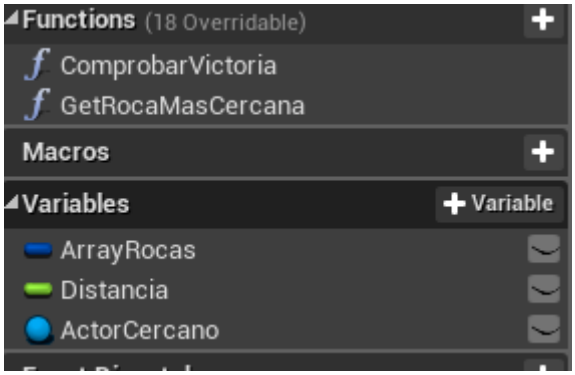

Fig. 55 Variables del Nivel

Ciertas funciones han sido programadas en los planos del nivel directamente, el motivo de esta decisión es que son funciones que utilizan datos relativos al nivel, o que afectan a todo el juego. Es aconsejable no recargar el nivel con funciones, especialmente si estas funciones pueden ser llevadas a otra clase, ya que el nivel es el que se encarga de calcular en tiempo de ejecución las luces dinámicas, posiciones de los objetos y todo lo relativo al sistema de físicas e iluminación.

Para la funcionalidad del nivel, se ha decidido crear funciones. Unreal te deja crear el plano de una función, y luego llamar a esa función como si fuera un nodo. Desde la función se puedes especificar los parámetros que recibe la función, así como aquellos que devuelve. También se han instanciado variables para dichas funciones.

Se han creado dos funciones, Comprobar Victoria y GetRocaMasCercana.

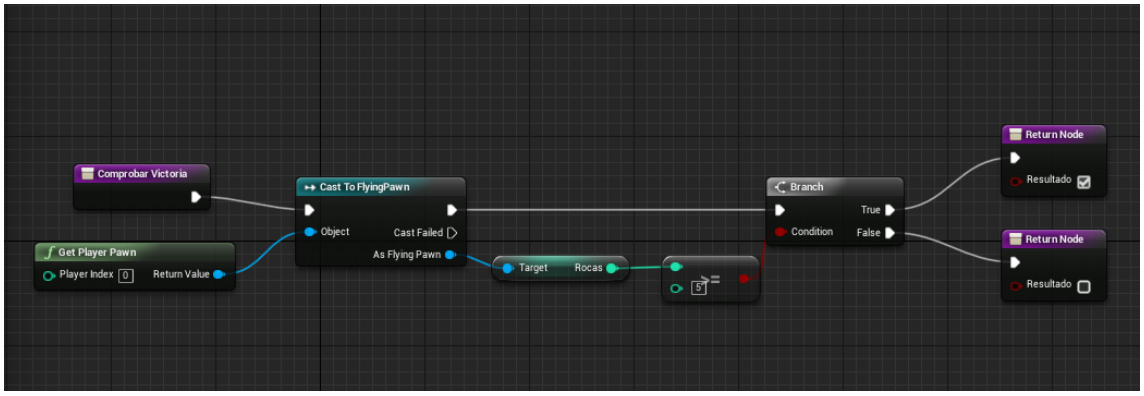

Fig. 56 Función Comprobar Victoria

ComprobarVictoria es la más sencilla de las dos, usando el nodo GetPlayerPawn consigue la referencia al peón que está poseyendo en dicho momento el jugador, se coge el índice de jugador 0 dado que es un juego para una sola persona, y no hay más jugadores. Con el peón, accede al atributo Rocas, y comprueba si se han recogido las 5 rocas, en caso afirmativo devuelve un booleano a True, en caso negativo devuelve False.

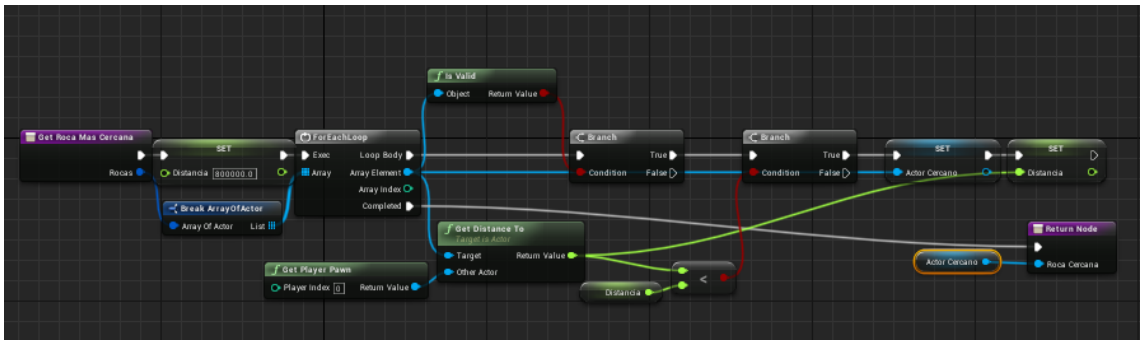

Fig. 57 GetRocaMasCercana

GetRocaMasCercana es más complicada. Lo que hace esta función es calcular la roca más cercana al jugador. Para ello, primero hace la variable distancia más grande que el mapa, para asegurar que siempre encontrara algo más cerca. Después, usa el array en el que se guardan las rocas para iterar, aunque primero tiene que hacer uso de BreakArrayOfActor, dado que el objeto array solo tiene la referencia al array, y es necesario usar ese nodo para conseguir la referencia a la lista que contiene el array.

Con el nodo ForEachLoop se itera por la lista, si la distancia entre la roca que se está iterando es más pequeña que la distancia mínima encontrada, se asigna esa roca como ActorCercano, y se pone esa distancia como nueva distancia mínima. El nodo de IsValid antes de asignar estos valores se asegura de que la roca este instanciada, y no haya sido ya destruida. Cuando se completa el bucle, se devuelve la roca más cercana.

Además de estas dos funciones hay 4 eventos en la pantalla. El evento de pausa, el de comenzar el juego, el de la condición de victoria y el de activar el sonar.

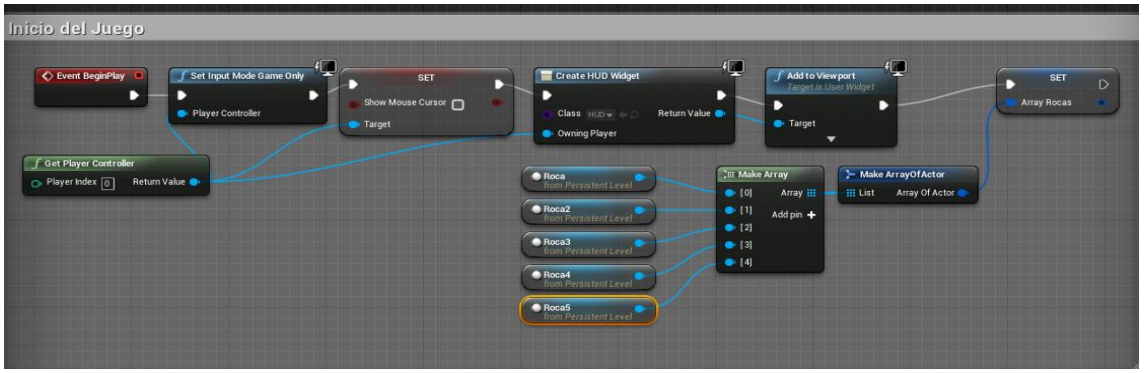

Fig. 58 Evento BeginPlay

El evento BeginPlay se dispara cuando el nivel es cargado. Se encarga de tres cosas, mediante los nodos SetInputModeGameOnly, y Set ShowMouseCursor, oculta el ratón del juego y da el control del juego al jugador, haciendo que las entradas no afecten a los menús. El nodo Create Hud widget junto al nodo AddToViewport hace que se muestre el HUD en la pantalla. Por último, se obtienen las referencias a las rocas del nivel y se instancia el array.

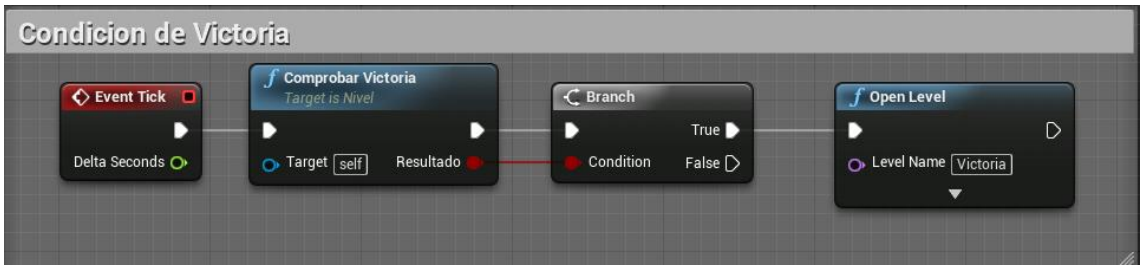

Fig. 59 Condición de Victoria.

En cada fotograma, con el nodo EventTick, se llama a la función comprobar victoria, si esta devuelve True, se abre el nivel de Victoria, que nos mostrará la pantalla apropiada.

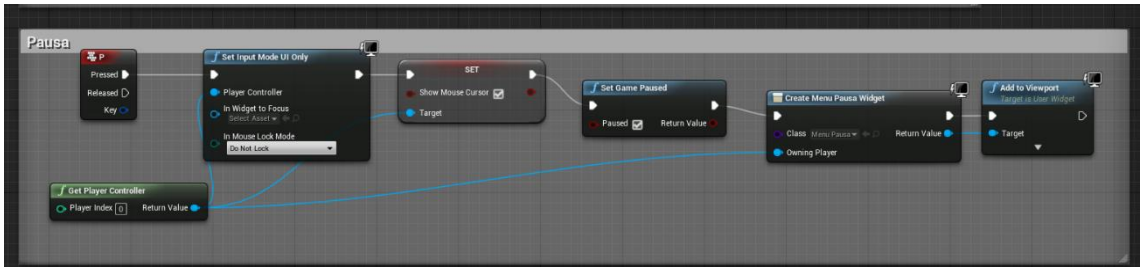

Fig. 60 Evento de Pausa

Cuando se detecta que se pulsa la tecla P, se usan los nodos SetInputModeUIOnly y set ShowMouseCursor para que se muestre el ratón y el peón ignore las órdenes del jugador. El nodo SetGamePaused se asegura de que todo el sistema de nivel se pausa, mejorando el rendimiento durante la pausa. Por último, se muestra el menú de pausa.

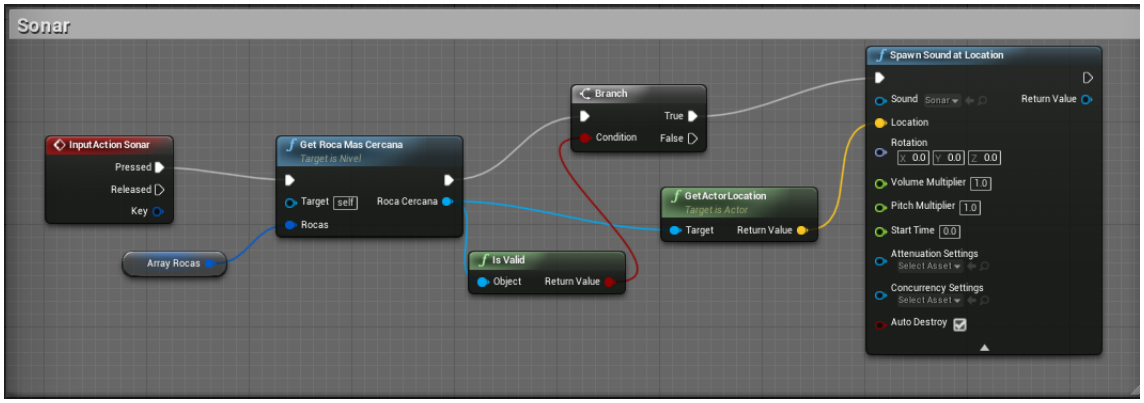

Fig. 61 Evento de llamada al Sonar

Cuando se detecta el control del Sonar, se llama a la función Get RocaMasCercana, después se instancia en la posición de la roca el sonido de sonar, modificado en Unreal para que use atenuación espacial para poder posicionarlo auditivamente.

Funcionalidad de las interfaces de usuario.

Existen ciertas cosas que hay que programar en las interfaces de usuario, por ejemplo, las acciones de los botones, o la lectura de la pantalla.

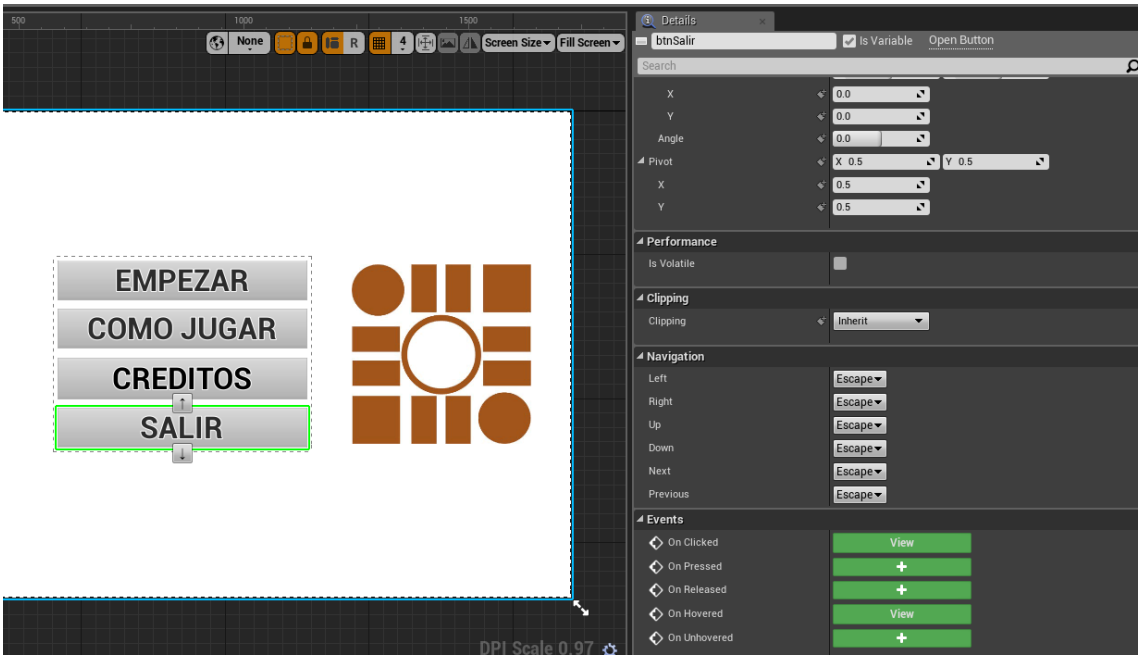

Fig.62 Eventos de los botones, en verde en la esquina inferior derecha

Lo primero siempre es crear las interfaces de usuario, como ya se ha visto antes con el nodo SetInputModeUIOnly para que se puedan usar los eventos de la interfaz y con los nodos CreateWidget y AddToViewport para mostrar el menú en la pantalla.

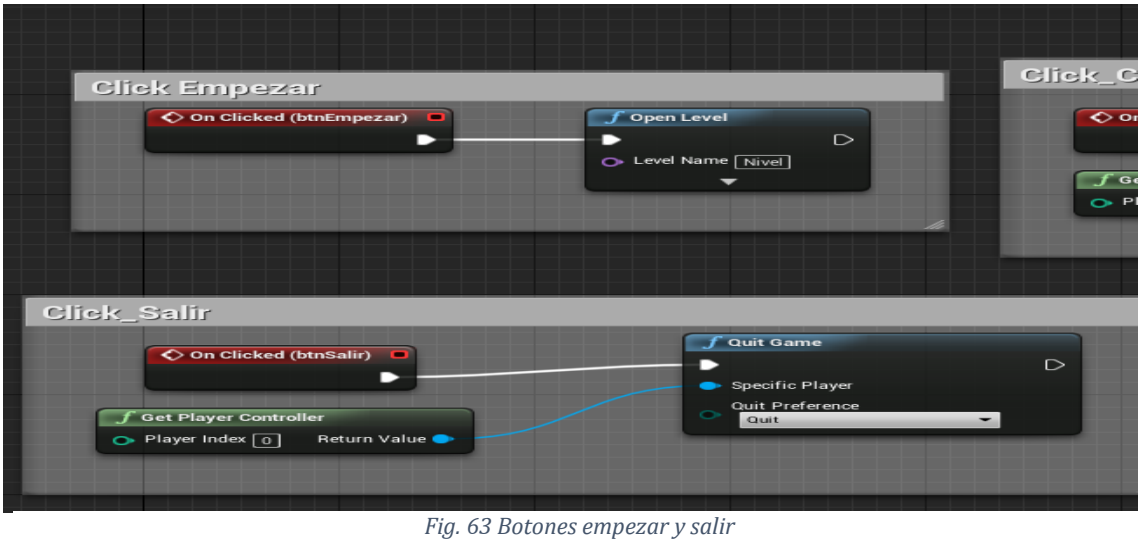

Fig. 63 Botones empezar y salir

Las funcionalidades de los botones se agregan mediante eventos que disparan esto, para añadir el evento, solo hay que ver las propiedades del botón desde la vista de diseñador. Si hacemos clic sobre el botón con el símbolo positivo, se abre el visor de eventos del widget y se crea el evento del botón.

Los botones de empezar y salir son sencillos. Empezar llama al nodo Open Level para abrir el nivel de juego. Salir usa el nodo Quit Game para sacar al jugador del juego. La funcionalidad de salir es la misma en todos los botones de Salir que tiene el juego. Desde el menú de pausa del juego también se hace uso del nodo OpenLevel para volver al menú principal.

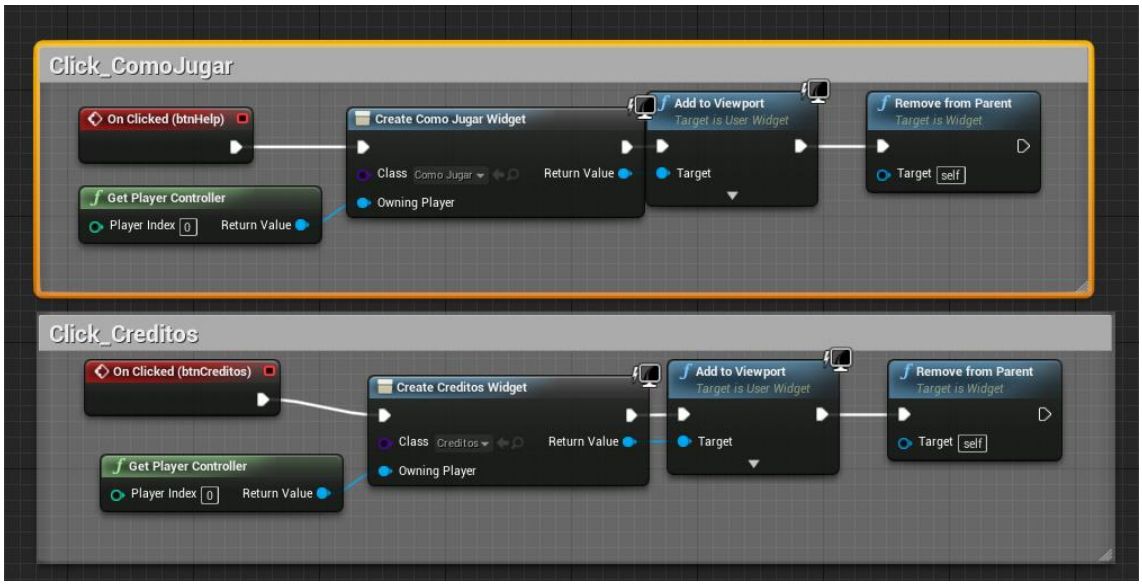

Fig. 64 Botones de como jugar y creditos.

Los botones de Como Jugar y créditos abren los widgets y los muestran, pero después se quitan ellos de la pantalla con el nodo Remove From Parent. Esta misma funcionalidad es la que usan la mayoría de los botones que cambian entre interfaces.

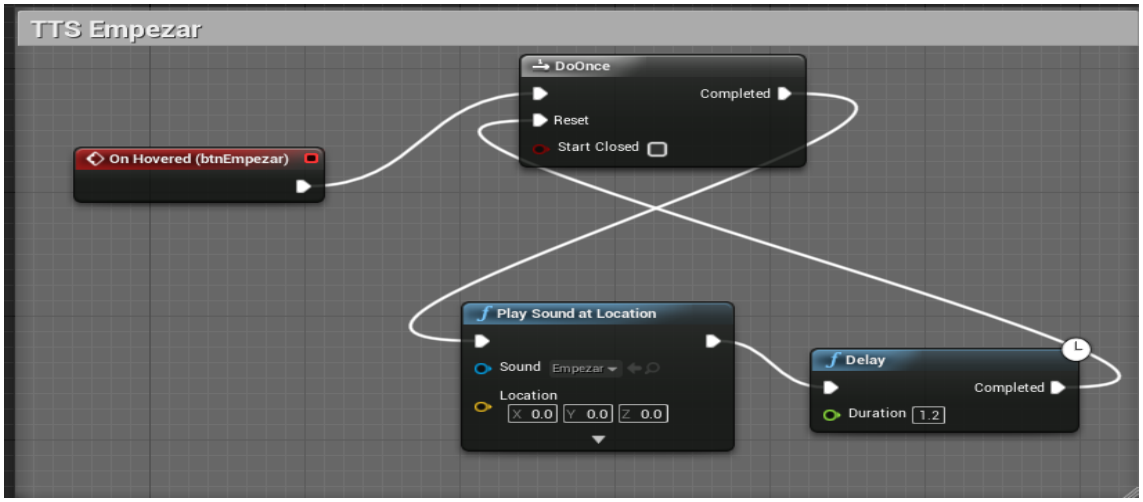

Fig. 65 Text to Speech del botón empezar.

Unreal Engine no es compatible con lectores de pantalla, ya que la mayoría de estos usan metadatos de la aplicación y Unreal no guarda estos metadatos. Por ello, se ha optado por programar un sistema de lectura de pantalla dentro de los menús más importantes del juego. En el menú principal todos los botones tienen la funcionalidad que se muestra en la

figura 64, usando el nodo DoOnce y OnHovered para que cuando el ratón pase por encima del botón se reproduzca un sonido que lea la función de ese botón.

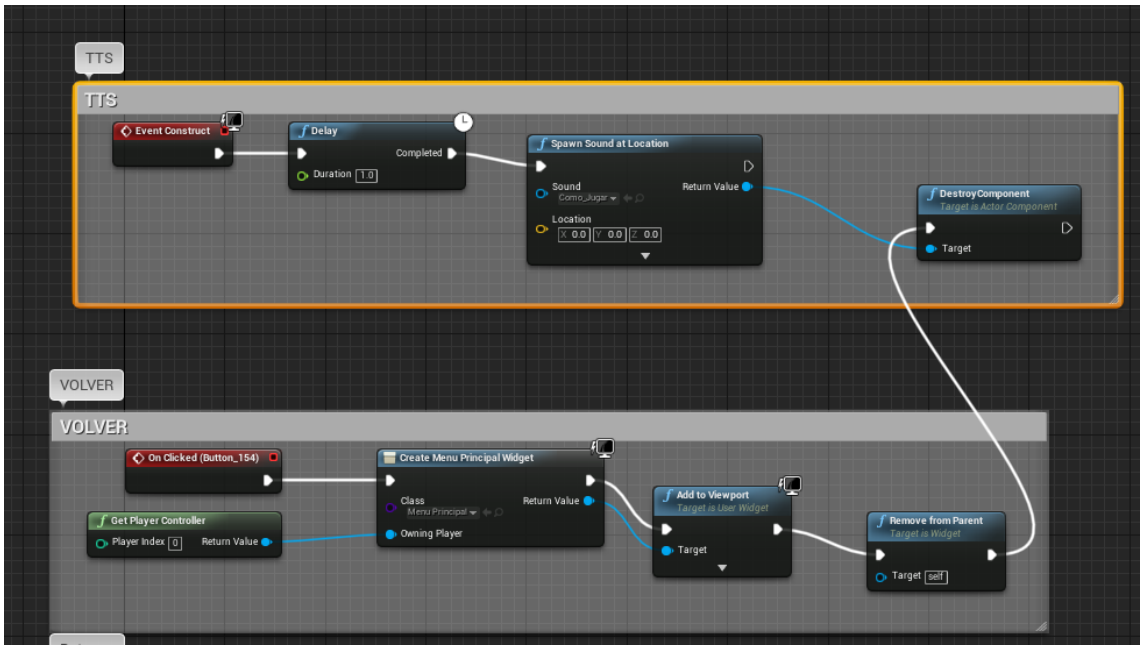

Fig. 66 Text To Speech del menú de Cómo Jugar

De forma similar, en el menú de Cómo Jugar se usa la misma lectura de los controles de la pantalla, sin embargo, esta vez se usa el nodo Spawn Sound at Location para tener la referencia del sonido, y si se pulsa el botón volver y se sale de este menú poder parar el sonido.

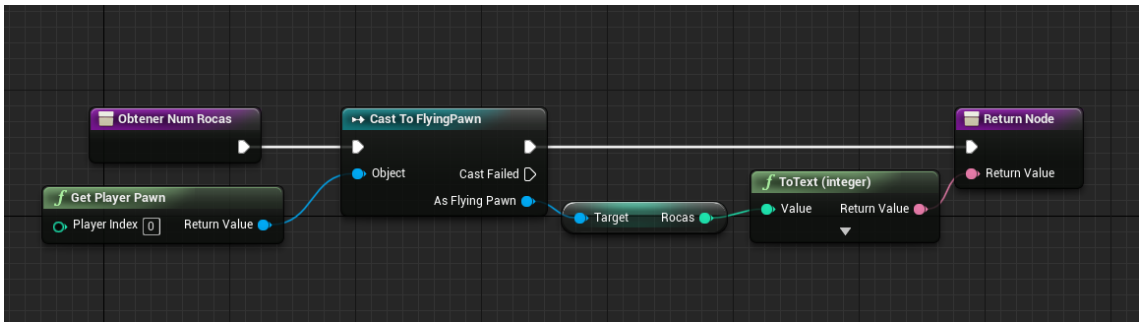

Fig. 67 Función Obtener Num Rocas para el HUD

Por último, señalar la función de Obtener Num Rocas, que es la función que tiene vinculada el texto del HUD, de tal forma que ese texto se actualiza cuando la función devuelve un valor distinto. La función simplemente muestra la variable Rocas que tiene el peón del jugador.

v. Análisis del cumplimiento de los requisitos.

En este apartado se analizará si se han cubierto de manera correcta todos los requisitos funcionales del prototipo, también se detallarán los problemas que se han encontrado durante el desarrollo de dicho prototipo.

<u>Cumplimiento de los requisitos.</u>

Requisito Funcional	Completo	Realización
RF01. El juego ha de iniciarse en un Menú Principal	SI	Esto se ha desarrollado con la realización de un plano de nivel que muestre un menú principal nada más abrirse el juego.
RF02. Ha de poderse salir del juego en cualquier momento	SI	En todo momento se puede ir a un menú o a una opción que permita cerrar el juego
RF03. Ha de existir una pantalla explicando todos los controles del juego	SI	Se ha diseñado una pantalla que explica Como Jugar
RF04. El objetivo del juego ha de ser recoger objetos en un entorno	SI	El objetivo del juego es recoger cinco rocas
RF05. El juego acabará una vez se hayan recogido todos los objetos	SI	Hay un evento que se desata cuando se cumple esta condición de victoria, ese evento cambia el nivel del juego a una pantalla de victoria que permite cerrar el juego
RF06. El juego ha de poder jugarse en su totalidad solo con el ratón o solo con el teclado	NO	La interfaz de usuario no es compatible con el teclado, aunque durante el nivel de juego si se puede jugar sólo con uno de los dos. Se explicará este problema en la siguiente sección del documento.
RF07. El juego ha de tener pistas sonoras que indiquen cuando se ha recogido un objeto	SI	El juego dispone de un sonido que se reproduce cuando se recoge una de las rocas.
RF08. El juego ha de tener pistas sonoras para indicar los objetos a coleccionar cercanos	SI	Cuando estás a una distancia cercana al objeto, se reproduce un sonido característico que indica que hay un objeto cerca.
RF09. El juego ha de tener pistas visuales que indiquen los objetos a recoger.	SI	Los objetos que deben ser recogidos son completamente blancos y reflejan el brillo de la iluminación completamente, destacando sobre el resto.
RF10. El juego ha de tener pistas sonoras que indiquen cuando se ha recogido un objeto.	SI	Cuando se recoge una roca se reproduce un sonido característico.
RF11. El juego ha de poder pausarse en cualquier momento	SI	Se puede pasar el juego pulsando la tecla P.
RF12. El juego ha de tener pistas visuales que indiquen cuando se ha recogido un objeto	SI	Cuando se recoge una de las rocas, dicha roca explota.
RF13. El juego ha de tener una pantalla de victoria que se muestre cuando se han recogido todos los objetos	SI	Se ha implementado una pantalla que indica al jugador su victoria cuando se alcanza la condición de victoria.

RF14. El juego ha de mostrar en todo momento cuantos objetos se han recogido	SI	El HUD se encarga de dar dicha información de forma actualizada
RF15. El juego ha de mostrar en todo momento cuantos objetos hay en total	SI	El HUD se encarga de dar dicha información
RF16. El juego ha de mostrar en todo momento el objetivo del juego	SI	El HUD se encarga de dar dicha información
RF17. El juego a detener una forma de localizar los objetos de forma auditiva	SI	El control de Sonar sirve para localizar los objetos de dicha forma
RF18. El menú principal y el de controles deben tener alguna forma de lectura de pantalla	SI	Se han implementado funciones propias de lectura de pantalla en ambos menús.

vi. Desafíos en el desarrollo.

A lo largo del proyecto se han encontrado diversos problemas de desarrollo que no se han podido resolver, por lo que se ha optado por vías alternativas. En este apartado se explicarán dichos problemas y soluciones.

1. El prototipo cambió de diseño una vez comenzado el desarrollo.
   La idea original del prototipo era un laberinto que también hubiera que explorar para poder encontrar una serie de objetos coleccionables. El objetivo era ver cuanto podía mejorar la capacidad de orientación de un jugador aplicando técnicas de accesibilidad.
   Sin embargo, el desarrollo se encontró parado por problemas de iluminación. Unreal Engine no permite hacer un juego sin iluminación, para que cualquier objeto muestre su textura, debe tener una luz, ya sea estática o dinámica, apuntando al objeto. Así pues, era necesarias dos cosas: añadir una linterna para asegurarnos de que el jugador siempre pudiera ver, y poner luces estáticas que aseguraran que todo estuviera iluminado.
   Esto además no tendría por qué afectar al proyecto gracias a que Unreal permite hacer uso de luces precompiladas que no han de construirse en tiempo dinámico, pues forman parte del nivel. Son similares a las luces estáticas, pero en ellas no se calcula iluminación. Además, desactivando las sombras se podría mejorar aún más el rendimiento, pues en el laberinto no serian necesarias. Por desgracia, el ordenador sobre el que se ha realizado el trabajo no era capaz de manejar las luces suficientes como para poder crear el nivel, y ambas veces que se intentó, el proyecto de Unreal dejaba de funcionar y se volvía inaccesible. Por ello, se decidió rediseñar el juego y convertirlo en un juego de exploración en un mapa, con una sola luz global, que mejorara el rendimiento. Además, a la hora de calcular las luces, se uso un objeto de importancia de iluminación (LightMassImportanceVolume) que centra los esfuerzos de la iluminación en un

solo punto, en nuestro caso alrededor de la pirámide, mejorando así aún más el rendimiento.

2. No se ha encontrado manera de implementar el uso del teclado en la interfaz. El requisito funcional 6 no ha podido ser cumplido de manera satisfactoria. Aunque el nivel de juego si es controlable tanto con el ratón como con el teclado, no se puede interactuar con la interfaz con el teclado, más allá de pulsar espacio para iniciar el juego.
   Esto es un problema que presenta Unreal Engine en cualquier proyecto. El problema se debe a que existen dos modos de detectar la entrada Game Mode y UI Mode, o modo de juego y modo de interfaz. Esto decide que clase consume primero la entrada, en el modo de juego, el peón del jugador detecta la entrada, en el modo de interfaz, la interfaz detecta primero la entrada y si no la consume luego se la pasa al peón. Estos modos son combinables. Sin embargo, la interfaz nunca escucha un evento de teclado, solo escucha al ratón. Algunos SDK si que añaden forma de comunicar con la interfaz sin ratón, ya que esto es indispensable para una consola.
   Se han investigado medidas que tomar para hacer que la interfaz reciba la entrada del teclado, probando sugerencias encontradas en las comunidades de desarrolladores de Unreal. Estos esfuerzos no han dado fruto, pues no se ha conseguido encontrar ninguna solución a tiempo para este problema que funcione correctamente.

3. No se han podido desarrollar controles personalizables. Como ya se menciona en el apartado de desarrollo, hasta hace escasas versiones Unreal Engine no permitía el cambio de las entradas que detectaba el controlador. Teóricamente es posible cambiar estas entradas en la versión del motor en la que se ha desarrollado el prototipo, sin embargo, probablemente por la novedad de la funcionalidad, no se ha conseguido implementar, pues no funcionaba de forma correcta.

4. No se ha podido realizar el proyecto con la calidad deseada. Unreal Engine es una herramienta muy flexible, que, de manera general, permite la personalización de prácticamente cualquier característica del motor. Esto se debe a que el sistema de Blueprints se complementa con C++, y siempre se puede acceder al código para escribir funciones propias. Sin embargo, el motor también es delicado, las funciones propias deben ser compatibles con el motor, y requieren gran conocimiento de la tecnología de Unreal y un esfuerzo temporal costoso. Además, la flexibilidad de Unreal respecto a diseño de imágenes, texturas, modelos, sonidos, etc. Exige conocimientos en gran cantidad de disciplinas distintas. Unreal es, claramente, un motor pensado y diseñado para equipos de desarrollo multidisciplinares y de tamaño medio-grande. Desarrollar un juego completo en Unreal con un solo desarrollador es una ardua tarea que exige dedicación, conocimiento de varias disciplinas, y un gran plazo de desarrollo. Por desgracia, nuestro equipo no dispone de conocimientos en muchas de estas disciplinas. Esto, junto a la limitación temporal, no ha permitido desarrollar el prototipo que

se deseaba, quedando ciertas funcionalidades y capacidades como trabajo futuro, del que se hablará más adelante.

b. Rúbrica de accesibilidad.

La rúbrica es uno de los objetivos más importantes del proyecto. En este apartado se aportará la información acerca de la rúbrica, su propósito y cómo usarla.

i. Diferencias entre la rúbrica y la lista de verificación de *Game Accesibility Guidelines* y objetivo de nuestra rúbrica

Game Accesibility Guidelines [14] es un documento colaborativo creado por un grupo de estudios de desarrollo de videojuegos, especialistas en accesibilidad y académicos. El objetivo del documento es exponer soluciones de accesibilidad orientadas a desarrolladores, con el objetivo de hacer la accesibilidad un requerimiento más común hoy en día.

Game Accesibility Guidelines es un centro de recursos que han sido de gran ayuda a la hora de desarrollar este trabajo. Su clasificación de las soluciones en grados de accesibilidad Básicos, Intermedios y Avanzados además de divididos entre los diferentes tipos es de gran ayuda. Además, se puede contactar con la organización para hacer propuestas y buscar soluciones. Uno de los discursos de los que disponen, es una lista de verificación, descargable desde su página web.

Guideline	Relevant to mechanic?	Implemented?	Notes
	("no" automatically greys out row)		
Motor			
(Control / mobility)			
Basic			
Allow controls to be remapped / reconfigured			
Ensure controls are as simple as possible, or provide a simpler alternative			
Ensure that all areas of the user interface can be accessed using the same input method as the gameplay			
Include an option to adjust the sensitivity of controls			
Ensure interactive elements / virtual controls are large and well spaced, particularly on small or touch screens			
Intermediate			
Support more than one input device			
Make interactive elements that require accuracy (eg. cursor/touch controlled menu options) stationary			
Ensure that multiple simultaneous actions (eg. click/drag or swipe) are not required, and included only as a supplementary / alternative input method			
Ensure that all key actions can be carried out by digital controls (pad / keys / presses), with more complex input (eg. analogue, speech, gesture) not required, and included only as supplementary / alternative input methods			
Include an option to adjust the game speed			
Avoid repeated inputs (button-mashing/quick time events)			
If producing a PC game, support windowed mode for compatibility with overlaid virtual keyboards			
Avoid / provide alternatives to requiring buttons to be held down			
Allow interfaces to be rearranged			
Allow interfaces to be resized			

Fig. 68 Lista de Verificación de Game Accesibility Guidelines[F10]

Dicha lista de verificación es similar a la rúbrica elaborada, sin embargo, presenta diferencias lo suficientemente sustanciales como para justificar la existencia de la rúbrica propuesta.

La lista de verificación de Game Accesibility Guidelines sólo anima a conocer diferentes soluciones, implementando aquellas que se piensen relevantes para la mecánica o el juego a desarrollar. Nuestra rúbrica, presenta distintas soluciones de accesibilidad también, pero

categorizadas de manera diferente, y busca revelar de manera aproximada cómo de accesible es un videojuego.

Somos conscientes de que la accesibilidad es, en última instancia, un requisito no funcional, y no se puede cuantificar de manera sencilla. Existen soluciones que impactan en la accesibilidad de un perfil en mayor medida que otras, y existen ciertas soluciones que pueden ayudar a varios perfiles al mismo tiempo. Además, al igual que suele suceder con la eficiencia o la seguridad de cualquier producto software, todo producto siempre puede ser más accesible.

Por ello, el objetivo de nuestra rúbrica es dar una medida aproximada de las soluciones tomadas, en que perfiles se necesita refuerzo, y que medidas se pueden implementar aún. Es nuestra opinión que, mostrando qué medidas se han implementado, y cómo pueden ser mejoradas, se anima a los desarrolladores a intentar mejorar.

ii. Creación y Uso de la Rúbrica.

Nuestra rúbrica se ha creado con la herramienta *Microsoft Excel*, ya que dicha herramienta permite fácilmente la creación de tablas, listados de valores y cálculos de medias que se necesitan para la creación de la rúbrica. El funcionamiento es sencillo, en nuestra tabla se muestran las columnas que pueden ser apreciadas en la figura 68. Estas columnas son: Perfil, Solución, Mejora, Implementado, Resultado y Total.

PERFIL	SOLUCIÓN	MEJORA	IMPLEMENTADO	RESULTADO	TOTAL
AUDITIVO	Existen subtitulos		SI	21,429%	
		Los subtitulos tienen un tamaño minimo de 46 pixeles.	NO		
		Es posible cambiar el tamaño de los subtitulos.			
		Los subtitulos contrastan con el juego o con un fondo.			
		Se puede cambiar el color de los subtitulos o del fondo.			
		Los subtitulos se organizan en frases cortas.			
		Los subtitulos muestran el nombre del personaje que habla.	NO		
		Los subtitulos son totales.			
	Existen pistas visuales				
		Existen indicadores visuales de los objetos con los que interactuar			

Adrian:
Si los subtitulos siempre contrastan con el fondo

Fig. 69 Rúbrica de Accesibilidad

- Perfil muestra a qué perfil pertenecen las soluciones mostradas.
- Solución muestra una de las soluciones que existen para mejorar la accesibilidad en ese perfil.
- Mejora indica las características que mejoran la solución a implementar.
- Implementado sólo permite dos valores ya que el resto están restringidos mediante una validación de datos (desde la barra de datos -> validación de datos.) Los valores permitidos son SI o NO.
- La columna de Resultado es automática, y muestra el resultado de una media entre las soluciones implementadas y no implementadas en dicho perfil. Esta media se calcula mediante una columna oculta que otorga 2 puntos por cada solución implementada, y 1 punto por cada mejora implementada en esa solución.
- La columna Total simplemente hace unas medias de los resultados alcanzados en cada uno de los perfiles.

Cabe destacar que, si una solución no es necesaria en el desarrollo de un prototipo, no nos parece justo “castigar” la media por una mejora que no se puede implementar por no

ser relevante (por ejemplo, un juego sin diálogos no necesita subtítulos). Por ello, ciertas columnas tienen un comentario que explican cuando no es necesario implementar dicha solución, si no es necesario, se puede marcar SI para que cuente como implementada.

La rúbrica marca en rojo todas las soluciones no implementadas, para recordar al desarrollador qué soluciones se pueden implementar aún. También se marca en rojo el resultado de un perfil si éste alcanza menos de un 60% de las soluciones implementadas, pues consideramos que es necesario prestarle más atención a dicho perfil si no se ha alcanzado dicha puntuación.

5. Conclusiones del Proyecto y trabajo futuro.

Durante la realización de este proyecto se han adquirido conocimientos en diferentes áreas. Se ha aprendido acerca de la accesibilidad en el mundo del entretenimiento y en productos software y se ha investigado y aprendido sobre el desarrollo de videojuegos en entornos 3D. Por último, también se ha aprendido sobre el desarrollo con el motor de videojuegos Unreal Engine.

En nuestra opinión la accesibilidad es uno de los requisitos que puede tener cualquier tipo de producto software que más afecte a su funcionalidad y que más impacto produzca sobre el resultado final. Es innegable que mejora la calidad del producto, además de, en la mayoría de los casos, ser económica de desarrollar. También creemos que existe un componente moral en la accesibilidad y que es un deber hacer todo producto tan accesible como sea posible. Existe una forma de conseguir que la accesibilidad se implemente en cualquier producto software de manera correcta y eficiente, y es tener en cuenta la accesibilidad desde la primera fase de diseño del producto.

Por desgracia, también se ha aprendido que la accesibilidad es, actualmente, algo que no se suele tener en cuenta a la hora de desarrollar videojuegos en la industria especializada. No existe un estándar en el sector para el desarrollo de la accesibilidad, siendo difícil encontrar recursos y conocimientos sobre qué soluciones son necesarias, cómo han de abordarse y qué consecuencias implican dichas soluciones. Existen muchos videojuegos que implementan ciertas opciones de accesibilidad, si bien en la mayoría de los casos suelen ser escasas y, en ocasiones, están pobremente diseñadas. Sin embargo, se ve una tendencia creciente a preocuparse por la accesibilidad en los videojuegos, con grandes empresas como *Ubisoft* intentando mejorar las soluciones que ya tienen implementadas, y organizaciones como *Game Accesibility Guidelines* preocupándose por crear centros de recursos y soluciones que puedan aprovechar los desarrolladores.

Unreal Engine es uno de los motores más potentes y más utilizados en la industria, especialmente después del éxito del videojuego *Fortnite* creado por *Epic Games*, la misma empresa desarrolladora del motor. Sin embargo, Unreal Engine es reflejo de la industria, y cuenta con escasas opciones de accesibilidad por defecto. Cierto es que siempre se pueden adquirir plug-ins y soluciones similares que permitan trabajar con la accesibilidad de forma más sencilla. También es cierto que el motor permite la modificación de casi todas sus cualidades a través de código C++, sin embargo, no ofrece documentación ni recursos, y realizar esto exige conocimientos extensos acerca del motor. Además, cuenta con otro

problema a la hora de desarrollar juegos de manera independiente, Unreal Engine está pensado para trabajar en un equipo de muchas personas con roles variados. Desarrollar cualquier videojuego en el motor exige conocimientos de programación, de sonido, de funciones físicas, y de diseño. Haciendo largo y costoso el desarrollo de cualquier proyecto por una sola persona, o un equipo reducido con limitaciones en sus disciplinas. Esto también daña la accesibilidad en dichas situaciones, a pesar de que las propuestas de accesibilidad para videojuegos suelen estar más presentes en los juegos independientes.

Existen ciertas ideas que no se han podido llevar a cabo en este proyecto, quedando abiertas como futuras líneas de investigación a explorar:

- Completar ciertas funciones de accesibilidad en el videojuego. Hacerlo compatible con varios controladores, añadir personalización de controles y el resto de los problemas expuestos durante el apartado de desafíos en el desarrollo.
- Actualizar la versión del motor sobre la que está diseñado el videojuego. Esto podrá ayudar en el desarrollo de la personalización de los controles, así como facilitar el desarrollo de nuevas características.
- Implementar un sistema de generación procedural de escenarios en el prototipo. Esto no ha podido ser desarrollado por las restricciones temporales del proyecto. Para realizar este trabajo habría que unificar modelos, en lugar de implementarlos por Unreal, y construir un algoritmo específico que ayudara a poblar el mapa de manera procedural con los nuevos modelos.
- Implementar distintos modos de juego en el prototipo. Aparte del modo de recolección actual, también podría haber distintos modos, como carreras contrarreloj o desafíos de precisión de vuelo. También debería añadirse un sistema de puntuación. Al igual que el punto anterior, no ha sido resuelto por la constricción temporal.
- Perfeccionar la rúbrica con el documento de *Game Accesibility Guidelines*. Modificar nuestra rúbrica de puntuación para que trabajara sobre la lista de verificación expuesta anteriormente, lo que permitiría un mayor acceso a los recursos y perfeccionaría el sistema de puntuación de la rúbrica, haciéndolo más preciso.

En cuanto a la rúbrica, hemos aprendido que la accesibilidad es un requisito no funcional, y, por tanto, es difícil de medir. Sin embargo, creemos que si es posible dar una aproximación del grado de accesibilidad en un videojuego o en un producto software. Consideramos la rúbrica útil por varios motivos. El primero, te permite comprobar para que perfiles has desarrollado mejor la accesibilidad, viendo en que se debe prestar mas atención. Lo segundo, también permite ver las técnicas mas comunes para mejorar la accesibilidad, y comprobar rápido si se han implementado o no. Esta rúbrica es sencilla, y como ya se ha explicado, puede mejorarse, pero su utilidad es amplia y presenta, en nuestra opinión, un gran potencial.

Por último, nos gustaría acabar este proyecto con una pequeña reflexión. Los videojuegos son para todos, por ello, deberíamos desarrollarlos para todos.

6. Bibliografía.

Referencias:

1. Forbes. “The best-selling videogames of 2017. (So far)” Disponible en: https://www.forbes.com/sites/erikkain/2017/12/16/the-best-selling-video-games-of-2017-so-far/#5751a24fab2b
2. Cruz roja. Tipos y grados de discapacidad. Disponible en: http://www.cruzroja.es/portal/page?_pageid=418,12398047&_dad=portal30&_schema=PORTAL30
3. Centers for Disease Control and Prevention (CDC). Tipos de pérdida auditiva. Disponible en: https://www.cdc.gov/ncbddd/spanish/hearingloss/types.html
4. Organización Mundial de la Salud. Sordera y pérdida de audición. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss
5. Organización Mundial de la Salud. Ceguera y discapacidad visual. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment
6. Naciones Unidas. Día mundial del Síndrome de Down. Disponible en: https://www.un.org/es/events/downsyndromeday/background.shtml
7. Plataforma Dislexia. La dislexia en España. Disponible en: http://plataformadislexia.org/la-dislexia-espana/
8. Cruz Roja. Discapacidades motoras. Disponible en: http://www.cruzroja.es/portal/page?_pageid=418,12391802&_dad=portal30&_schema=PORTAL30
9. Liz England. “The Door Problem”. Disponible en: http://www.lizengland.com/blog/2014/04/the-door-problem/
10. Ubisoft. How Ubisoft is putting the spotlight on accessibility. Disponible en: https://news.ubisoft.com/en-us/article/324314/how-ubisoft-is-putting-the-spotlight-on-accessibility
11. Variety. The Blind Masters of Fighting Games. Disponible en: https://variety.com/2018/gaming/features/blind-players-fighting-games-1202856319/
12. Brannon Zahand on Twitter. Disponible en: solo costo a Ubisoft 8.5 hombres/día completar los subtítulos de Assasins Creed Origins.
13. Wikipedia. Ray Casting. Disponible en: https://es.wikipedia.org/wiki/Ray_casting
14. Game Accessibility Guidelines. Disponible en: http://gameaccessibilityguidelines.com/

Figuras:

F1. Imágenes del Oído Humano: Disponible en: https://saberimagenes.com/oido-humano-estructura-partes-nombres/
F2. Tratamiento del Daltonismo. Disponible en: https://optometristas.org/?q=tratamiento-del-daltonismo
F3. Wikipedia. Sindrome de Down. Disponible en: https://es.wikipedia.org/wiki/S%C3%ADndrome_de_Down
F4. Studio Cypher. The Door Problem. Disponible en: http://studiocypher.com/the-door-problem
F5. Game Accesibilty Guidelines. Menú Accesible de Celeste. Disponible en: http://gameaccessibilityguidelines.com/celeste-assist-mode/
F6. Ian Hamilton on Twitter. Acerca de los subtítulos de Valve. Disponible en: https://twitter.com/ianhamilton_/status/1065175108531249152
F7. Reddit. Audio Visualizers Added to Fortnite. Disponible en: https://www.reddit.com/r/NintendoSwitch/comments/8y90vm/audio_visualizer_also_added_to_all_fortnite/
F8. Xataka. Cómo jugar a Videojuegos y sobrevivir siendo Daltónico. https://www.xataka.com/videojuegos/como-jugar-a-videojuegos-siendo-daltonico-y-sobrevivir-en-el-intento
F9. Fandom, Civilization Wiki. Hex. Disponible en: https://civilization.fandom.com/wiki/Hex
F10. Game accessibility guidelines. Lista de verificación de accesibilidad. Disponible en: http://gameaccessibilityguidelines.com/excel-checklist-download/

7. Anexos.

a) Resultados del prototipo en la rúbrica de accesibilidad.

PERFIL	SOLUCIÓN	MEJORA	IMPLEMENTADO	RESULTADO	TOTAL
AUDITIVO	Existen subtitulos		SI	85,714%	62,218%
		Los subtitulos tienen un tamaño minimo de 46 pixeles.	SI		
		Es posible cambiar el tamaño de los subtitulos.	SI		
		Los subtitulos contrastan con el juego o con un fondo.	SI		
		Se puede cambiar el color de los subtitulos o del fondo.	SI		
		Los subtitulos se organizan en frases cortas.	SI		
		Los subtitulos muestran el nombre del personaje que habla.	SI		
		Los subtitulos son totales.	NO		
	Existen pistas visuales		SI		
		Existen indicadores visuales de los objetos con los que interactuar	SI		
		Existen indicadores visuales de las acciones que evitar	SI		
VISUAL	Hay un modo de alto contraste		SI	60,000%	
		Existe un esquema de color para la Deuteranopia y la Deuteranomalía	SI		
		Existe un esquema de color para la Protanopia y la Protanomalía	SI		
		Existe un esquema de color para la Tritanopia y la Tritanomalía	SI		
		Se puede cambiar el brillo del juego	NO		
	El HUD esta simplificado mediante iconos		NO		
	Los colores del juego se pueden seleccionar		SI		
	Se puede cambiar el tamaño de los textos		NO		
	Se puede modificar el volumen del juego		NO		
		Los distintos volumenes del juego se pueden modificar por separado	NO		

	Compatibilidad con lectores de pantalla o TTS		SI		
	Existen pistas auditivas		SI		
		Existen indicadores auditivos de los objetos con los que interactuar	SI		
		Existen indicadores auditivos de las acciones que evitar	SI		
COGNITIVO	Siempre se muestra el objetivo actual		SI	63,158%	
		El objetivo, mejor explicado, puede consultarse	NO		
	Se puede pausar el juego		SI		
	Existe un modo de vista simplificada		SI		
	La tipografia del juego puede cambiarse		SI		
	El texto no avanza automaticamente.		SI		
	Existe un registro de los dialogos		NO		
	Existe un modo de tutorial / práctica		NO		
	El juego se puede guardar en cualquier momento		NO		
	Existe un sistema de guiado hacia tu objetivo.		SI		
MOTOR	Existe un esquema de control simplificado		SI	40,000%	
	Existe un sistema de control personalizado		NO		
	Existe compatibilidad con varios controladores		NO		
	Se puede modificar la dificultad del juego		NO		

	Existen formas de autocompletar los QTE		SI		

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá